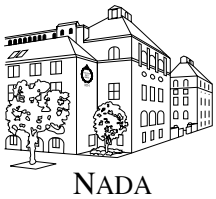
Web Based Booking System

Sara Hvatz

TRITA-NA-E03081



NADA

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Web Based Booking System

Sara Hvatz

TRITA-NA-E03081

Master's Thesis in Computer Science (20 credits)
within the First Degree Programme in Mathematics and Computer Science,
Stockholm University 2003
Supervisor at Nada was Kjell Lindqvist
Examiner was Lars Kjelldahl

Abstract

SaabTech Systems has in the past couple of years developed some products for the commercial market and the Web Based Booking System (WeBaBoS) is one of these products. The purpose of this product is to simplify the process for residents to book the domestic facilities, for example the laundry room, in their building complex. There are already similar booking systems on the market, but SaabTech Systems would like to produce its own prototype. The pilot system will be further evaluated by the tenant-owner's association *Sjötungan* in Tyresö.

The system includes internal clients in the form of computers with touch screens that are connected to an internal LAN, and external clients, which can be any computers with Internet access. An Internet server together with a database constitute the heart of the system. The server supplies the Web pages, and to provide dynamic and interactive web pages, system safety, and general software, the database includes all the important data such as resident data and facility constraints.

For the WeBaBoS project the Waterfall lifecycle has been chosen as a guideline for how to manage the project from the beginning to the end. Before developing the software, an efficient platform for this type of system had to be chosen. The programming languages Java or C++ were from the early stage considered to be the most suitable language to use, but after acquiring a better understanding of the system it was obvious that it would be better to use either the technique ASP, ASP.net or the language PHP. Whether an object-oriented, relational or object-relational database was most appropriate for the system was a question as well. Finally, after research and analysis, the platform Windows Web server IIS, the server-based Internet programming language PHP, and the relational database MySQL were chosen.

The finished prototype has been tested and approved by the customer, who already has plans for future additional functions.

Referat

Internetbaserat bokningssystem.

SaabTech Systems har under några år utvecklat produkter för den kommersiella marknaden och detta internetbaserade bokningssystem är en av dem. Syftet med bokningssystemet är att underlätta för människor att boka sina gemensamma objekt och lokaler i en bostadsförening. Det finns redan liknande produkter att köpa på marknaden men SaabTech Systems har ändå för avsikt att utveckla en prototyp. Prototypen kommer sedan att vara grund för pilotsystemet som ska testanvändas av bostadsrättsföreningen Sjötungan i Tyresö.

I systemet ingår det interna klienter i form av pc-datorer med pekskärm som är kopplade till ett LAN och externa klienter som praktiskt taget är vilken dator som helst som har internetuppkoppling. En internetserver och en databas utgör tillsammans systemets hjärta. Servern tillhandahåller webbsidorna och för att uppnå dynamiska och interaktiva webbsidor, systemsäkerhet och generell mjukvara finns alla viktiga data, sådant som till exempel medlemsdata och objektens restriktioner, lagrade i databasen.

För detta mjukvaruprojekt har Vattenfallsmodellen valts ut som en guide för hur projektet skulle organiseras från början till slut. Innan själva utvecklingen av mjukvaran skulle en effektiv plattform för detta system väljas. Java eller C++ var de programmeringsspråk som från början ansågs vara de lämpligaste att använda vid utvecklingen, men allteftersom insikten om systemets uppbyggnad och syften ökade stod det klart att programmeringstekniken ASP, ASP.net eller programspråket PHP var lämpligare alternativ. Huruvida antingen en objekt-orienterad, relations-, eller objekt-relationsdatabas var det bästa alternativet av databaser var också en fråga som undersöktes. Efter forskning och analysering blev plattformen Windows webbserver IIS, det serverbaserade internetspråket PHP och relationsdatabasen MySQL vald.

Den slutliga prototypen har genomgått tester av olika slag och blivit godkänd av kunden som redan har framtida planer på att utöka systemets funktionalitet.

Acknowledgements

This project is a Master's thesis in Computer Science at Stockholm University. Lars Kjelldahl is the examiner and Kjell Lindqvist is the supervisor, the important link between the school, SaabTech Systems and me.

The project has been accomplished at SaabTech Systems in Järfälla at the hardware department of the Common Technology division. The initiator of the thesis is Robert Frick whom I thank for showing true interest in the project. Thanks also to Mats Tholén for coming up with smart ideas in right moments, Ehsan Khan for research tips and "good cooperation", Thomas Ljungvall for "starting-up" help and "after five o'clock" support, and Michael Riordan for encouragement and language expertise.

I would also like to thank the rest of my colleagues in the "corridor" for fun and interesting conversations during lunchtime and coffee breaks.

Contents

1 Introduction 1
1.1 Problem 1
2 Software life cycle models 3
2.1 UR phase: user requirements definition 3
2.2 SR phase: software requirements definition 4
2.3 AD phase: architecture design 4
2.4 DD phase: detailed design and production 4
2.5 TR phase: transfer 5
2.6 OM phase: operations and maintenance 5
2.7 Life cycle approaches 5
2.8 The Waterfall Approach 6
2.8.1 The incremental delivery approach 7
2.8.2 The evolutionary development approach 8
2.8.3 Choice of approach for the WeBaBoS project 8
3 UR phase of the WeBaBoS project 9
3.1 Interview with the customer 9
3.2 Market research 9
4 SR and AD phase of the WeBaBoS project 11
4.1 Choosing the programming language 11
4.1.1 ASP vs. PHP vs. ASP.NET 12
4.1.2 Comparing PHP, ASP and ASP.NET 14
4.1.3 Conclusions 14
4.2 Choosing the database model 15
4.2.1 Relational databases 16
4.2.2 Relational structure 17
4.2.3 Constraints 18
4.2.4 Query languages 18
4.2.5 Object-Oriented databases 19
4.2.6 Object-oriented structure 19
4.2.7 OQL (Object Query Language) 20
4.2.8 Object-Relational 21
4.2.9 Object-relational structure 21
4.2.10 SQL3 22
4.2.11 Conclusions of the choice of database 22
4.3 User transactions 24
4.4 State chart of WeBaBoS 27
5 DD phase of the WeBaBoS project 28

5.1 Software environment 28
5.2 Difficulties during software development 28
6 TR phase of the WeBaBoS project 30
6.1 Requirement tests 30
6.2 Simultaneously booking test 30
6.3 Server and database crash tests 30
6.4 Web browser tests 31
6.5 GUI design tests 31
7 Results 32
7.1 The Waterfall model 32
7.2 PHP and MySQL 32
7.3 The WeBaBoS database structure 32
7.4 The GUIs design – Sunny Day Scenarios 35
7.5 The GUIs design – Rainy Day Scenarios 43
7.6 The card reader 43
7.7 The printer 44
7.8 Maximum performance 44
7.9 The Time plan and Limitations 44
8 Future thoughts 45
Definitions, acronyms and abbreviations 46
References 48
Appendix A 51
Appendix B 59
Appendix C 73
Appendix D 89

1 Introduction

SaabTech Systems AB is a Swedish company in the Saab Group. The company employs about 700 people and generates annual sales of about SEK 1.3 billion. The head office is in Järfälla (outside Stockholm), but the company also has offices in Härnösand, Uppsala, Västerås, and a subsidiary in Adelaide, Australia. SaabTech Systems has a 40-year tradition of developing and integrating Command and Control Systems for all branches of the armed forces and has sold systems in more than a dozen countries.

Since a couple of years the company has had ideas for developing some products for the commercial market and the *Web Based Booking System* (WeBaBoS) is one of these. The idea is to create an "Intelligent House" with technical solutions to simplify daily life. The hardware of the prototype for the WeBaBoS was developed by an undergraduate student and the next step was to develop the software to be integrated into the hardware.

1.1 Problem

The booking system that has been developed will be evaluated by a tenant-owner's association called *Sjötungan*. The association has 601 apartments and needs 64 terminals to place in the apartment complex entries. The complex includes several facilities, which are possible for the residents to book. A booking system can be very useful for a large complex such as this. It is convenient for the users when all of the facilities booking schedules are gathered in one place such as in a computer system. The requirements of the system have to some extent been tailored for *Sjötungan's* apartment complex. Some of the requirements are general but some are made to fit specifically their complex and their facility rules. The requirements include functionality, security and performance, and are gathered in the URD (Appendix A), but here are some of the main requirements listed:

General requirements:

- The system's internal terminals will be connected with a server on a LAN.
- The server will be connected to a database.
- The GUIs will be web based.
- The standard protocol TCP/IP will be used in network communication.
- All communication over the LAN will be cipher coded.
- The software will be as general as possible to facilitate additions of future functions.
- The software must be written in a programming language that works on most modern operating systems.

- The GUI must be designed in a way that people without computer experience can easily learn how to operate the system.

Specific requirements:

- To log in to the system the user must enter the card/member number and a personal code.
- A maximum of 100 users can be logged-in at the same time.
- The laundry rooms can be booked three hours per occasion and maximum three occasions per every three weeks period.
- The two gyms, the tennis court and the ping-pong room can be booked between 9.00 a.m. and 9.00 p.m., from one hour to a maximum of ten hours per allowable period. The period of how far ahead the facility can be booked depends on the facility.
- The sauna can be booked on the weekends between 9.00 a.m. and 9.00 p.m., one hour per family per day.
- The banqueting room can be booked for one hour up to one day.
- When the internal terminal is not used it will work as a general information site (automatically scrolling).
- If a user has tried to log in with an incorrect code three times the card/member number will be locked for X minutes and only the administrator may unlock the number before the X minutes have passed.
- When a booking has been made the user can choose whether he wants a printed receipt or not.
- If a logged-in user is idle for more than three minutes the user will automatically be logged out.
- A booking will remain in the database after the booking date has expired in order to retain some history of the bookings.

The requirements of the prototype are relatively straight forward, and all of them have been considered in the system design. The question was, how would the system be designed? Which programming languages, server, database, and operating system would be suitable for the system's purposes? What kind of user transaction management would be applied? How would the GUIs be designed so that everyone could use the system no matter what kind of previous knowledge of computers the end user has? In addition to the basic functions another requirement is that the code must be general enough to allow new functions to be easily added. SaabTech Systems already had some ideas of what kind of functions they might want to add, for example automatic door openings by letting a card reader read the member card, or placing a camera outside the building allowing the members to see the guest who rings on the house phone.

2 Software life cycle models

The first decision I had to make after realizing the purpose of the project was to decide a suitable software life cycle model for the project. So, what is a "life cycle" in this meaning? One can say that a software life cycle starts when a software product is conceived and ends when it is no longer available for use, i.e. it contains the whole of the development, operations and maintenance activities. General, structured models are necessary to make the software development process most efficient, especially if the project is large. A "life cycle model" structures project activities into "phases" and defines what activities occur in which phase. A "life cycle approach" is a combination of the basic phases of the life cycle model. The "approaches" can be designed in many different ways and every company or institution that develops software probably has their own approaches that best suit their business. However there are some models that are more frequently used than others. The European Space Agency (ESA) has a long tradition of large software projects and uses well prepared models and approaches, so when a model had to be selected for this project three of ESA's standard models were compared. They are different but every model includes these basic phases:

- UR phase – Definition of the user requirements
- SR phase – Definition of the software requirement
- AD phase – Definition of the architecture design
- DD phase – Detailed design and production of the code
- TR phase – Transfer of the software to operations
- OM phase – Operations and maintenance

ESA's own descriptions of the phases are described in [8].

2.1 UR phase: user requirements definition

The UR phase is the "problem definition phase" of a software project. In this phase the scope of the project is being defined and the requirements are captured in the URD. This can be done in different ways, for example by interviews, survey, or by building prototypes. The review of the URD should be done by the involved which are the end users, the software and hardware engineers, and the managers. The approved URD is the input to the SR phase.

Before the completion of the User Requirements Review, a Software Project Management Plan that outlines the whole project should be produced by the developer. This plan often contains a cost estimate for the project.

2.2 SR phase: software requirements definition

The SR phase is the "analysis" phase of a software project. A vital part of the analysis activity is the construction of a "model" describing what the software has to do, and not "how" to do it. The output of this phase is the Software Requirements Document (SRD), which as well must be reviewed formally by the concerned parties.

During the SR phase, the section of the Software Project Management Plan outlining the rest of the project must be updated.

2.3 AD phase: architecture design

The purpose of the AD phase is to define the structure of the software. The model constructed in the SR phase is the starting point. This model is transformed into the architecture design by allocating functions to the software components and defining the control and data flow between them.

This phase may involve several iterations of the design. Technically difficult or critical parts of the design have to be identified. The Architecture Design Document (ADD), which is the formal output of this phase, should be produced for every software project.

During the AD phase, a Software Project Management Plan outlining the rest of the project must be produced.

2.4 DD phase: detailed design and production

The purpose of the DD phase is to detail the design of the software, and to code, document and test it. The detailed Design Document (DDD) and the Software User Manual (SUM) should be produced concurrently with coding and testing. Initially, the DDD and SUM contain the sections corresponding to the top levels of the system. As the design progresses to lower levels, related subsections are added.

During this phase, unit, integration and system testing activities are performed according to verification plans established in the SR and AD phases. As well as these tests, there should be checks on the software quality.

After the outputs (code, DDD, SUM) have been formally reviewed the software can be declared ready for provisional acceptance testing.

2.5 TR phase: transfer

The purpose of this phase is to establish that the software fulfils the requirements laid in the URD. This is done by installing the software and conducting acceptance tests.

When the software has been demonstrated to provide the required capabilities, it can be provisionally accepted and operations started.

The Software Transfer Document (STD) must be produced during the TR phase to document the transfer of the software to the operations team.

2.6 OM phase: operations and maintenance

Once the software has entered into operation, it should be carefully monitored to confirm that it meets all the requirements defined in the URD. Some of the requirements, for example those for availability, may take a period of time to validate. When the software has passed all the acceptance tests, it can be finally accepted. After final acceptance, the software may be modified to correct errors undetected during earlier phases, or because new requirements arise. This is called “maintenance”.

The Project History Document (PHD) summarises the significant managerial information accumulated in the course of the project.

2.7 Life cycle approaches

The software life cycle model summarises the phases and activities. A life cycle approach, based upon this model, should be defined for any project, large or small.

This section defines three approaches that cover most of the needs of the ESA [8] but can be applied on other projects as well. In the diagrams, the phases have been reduced to boxes. Arrows connecting the boxes represent permitted transactions.

2.8 The Waterfall Approach

The "waterfall" approach, shown in Figure 1, is the simplest of the three approaches. The phases are entered one after another, as soon as the output of the current phase is produced. Although sometimes it is necessary to go back to a phase for updating or error correction. Delivery of the complete system occurs at a single milestone at the end of the TR phase. The approach allows the relationship between the customer and developer to be simple.

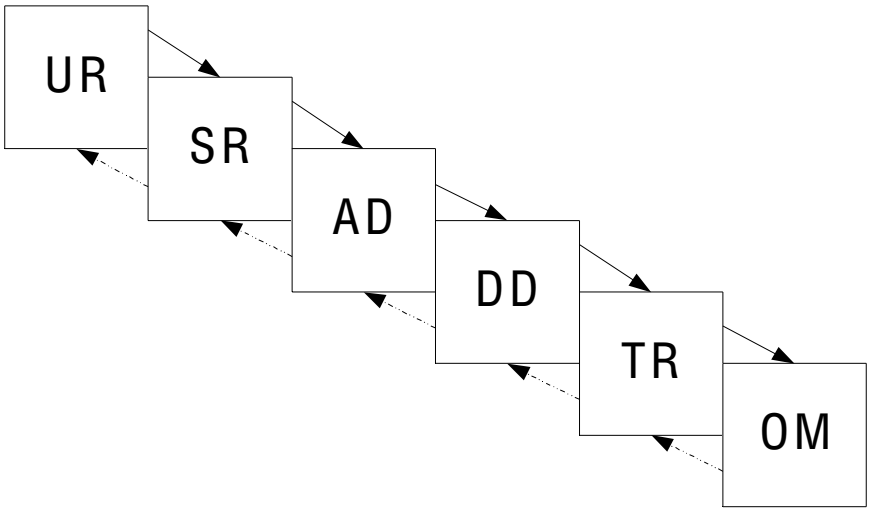


Figure 1. The waterfall approach. The dashed arrows represent updates in earlier phases.

2.8.1 The incremental delivery approach

The "incremental delivery" approach, shown in Figure 2, is characterised by splitting the DD, TR and OM phases into a number of more manageable units, once the complete architectural design has been defined. The software is delivered in several releases, each with increased functionality and capability. This approach is beneficial for large projects, where a single delivery would be impractical for reasons such as:

- certain functions may need to be in place before others can be effective;
- the size of the development team may necessitate subdivision of the project into a number of deliveries;
- the budget may only allow partial funding over a number of years.

In all cases, each deliverable should be usable, and provide a subset of the required capabilities.

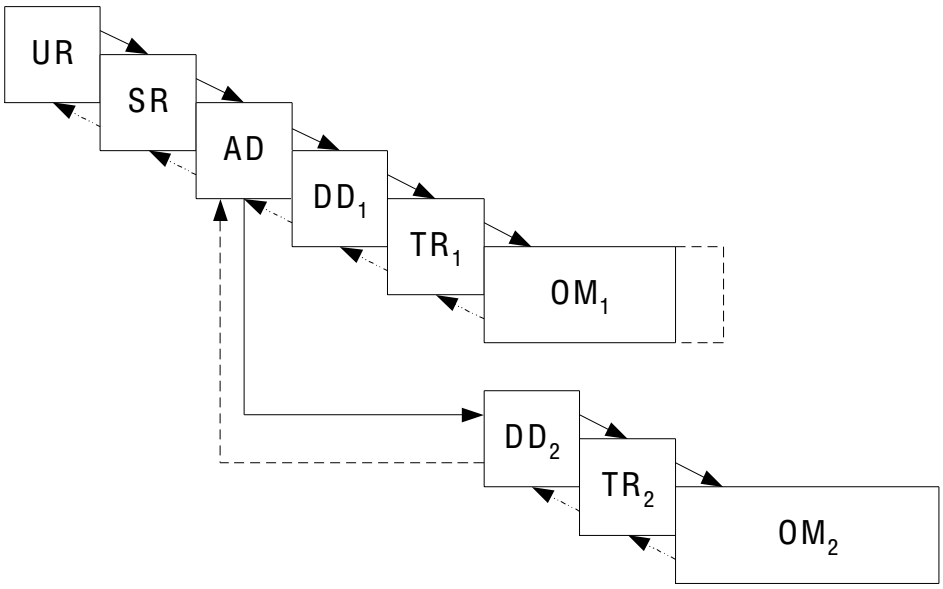


Figure 2. The incremental delivery approach.

2.8.2 The evolutionary development approach

The "evolutionary" approach, shown in Figure 3, is characterised by the planned development of multiple release. Each release incorporates the experience of earlier releases. The evolutionary approach may be used because for example:

- some user experience is required to refine and complete the requirements (shown by the dashed line within the OM boxes);
- some parts of the implementation may depend on the availability of future technology;
- some new user requirements are anticipated but not yet known;
- some requirements may be significantly more difficult to meet than others, and it is decided not to allow them to delay a usable delivery.

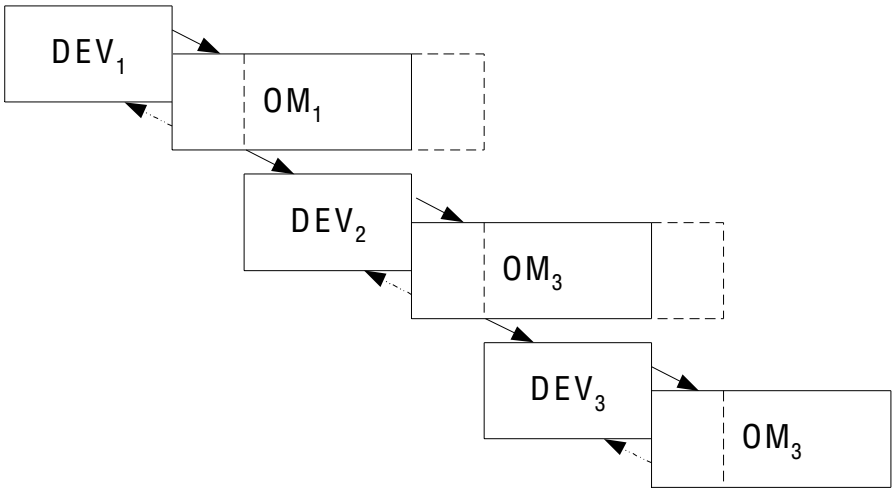


Figure 3. The evolutionary development approach. The dashed lines within the OM boxes represent refinements and completions. The dashed extensions to the boxes show that some overlap of OM phases will occur until each new delivery is finally accepted.

2.8.3 Choice of approach for the WeBaBoS project

For the WeBaBoS project it was from early stage decided that:

- the requirements from the customer would be precise and given all at once,
- there would be a full delivery, which means that the prototype would be delivered all at once and without any further undertakings,
- the system was small with few components.

Therefore the simplest one, the Waterfall Approach, was judged to be the most appropriate in this case.

3 UR phase of the WeBaBoS project

The first phase in the Waterfall model is the UR phase, or the "problem definition phase". To find out about the requirements I interviewed the customer, in this case Robert Frick, who was the initiator of the project. After the interview a time plan was set out for the project as well as a deadline of when the prototype was supposed to be delivered.

3.1 Interview with the customer

The interview led to the User Requirement Document (Appendix A) in which all of the customer's requirements were presented in a clear and unambiguous way. Before preceding to the next step the customer had to approve the document so there would be no misunderstandings after finishing the prototype.

3.2 Market research

When it was clear what kind of system the customer was looking for it was natural to do market research to see if such a product or a similar one already existed. In fact there were some products that fulfilled the requirements of the customer.

Frontyard AB is an international company with a Swedish base in Gothenburg. The company develops and sells software products and services to simplify everyday communication and their main markets are Intelligent Living and Video Conference. One of their products is the system called *Home Gateway System*. To make it convenient for the users the terminals are placed inside the apartments. The GUIs have a nice, colourful design and are even suitable for children. The customer can decide which functions to be available but some of the basic functions include: booking of the laundry room and the banqueting room, entry phone with camera, weather information, a calendar, and a digital photo album. It is also possible to add some more sophisticated functions like a fire detector, and automatic connection to the emergency service centre.

Electrolux Wascator AB is a company included in *The Electrolux Group* that is mostly known for their domestic appliances, for example, dish washing machines, vacuum cleaners, and washing machines. Electrolux Wascator is focused on laundry equipment and one of their products is the *Electrolux-Wascator Booking System*. It includes all main functions but also many other different functions. The system can be integrated with their passage system called *Aptus*. *Aptus* can, among other functions, control the entry phone, the elevator, and electronic name signs on the doors and outside the building.

Both of these companies have products that very well would satisfy the requirements (except that of a printed receipt) of the WeBaBoS. So why was the prototype still made after finding those alternatives? It is simple. *SaabTech Systems* is interested to try new markets and new techniques. In addition, since they already had a customer who was interested in buying this product, this was a good opportunity.

4 SR and AD phase of the WeBaBoS project

When the URD had been reviewed and approved it was time to enter the AD phase. Actually, according to the model, the SR phase is prior to the AD phase but since this project was a rather small one the most important parts from the SR phase, for example defining the use cases, were embedded into the AD phase.

The AD phase resulted in the ADD (Appendix B) that describes the system's different entities and how they interact. But to be able to design the system in detail some analysis had to be done.

4.1 Choosing the programming language

During the system design process it was necessary to decide which programming languages and scripts would be best for this particular system. In the URD I had mentioned that Java would be used but the more I understood of the system design the more I realized that there were other languages that might work better for the system. In order to choose the language, the requirements of functionality, security, and performance had to be considered.

HTML had to be used since it is the essential layout script for web pages. Obviously it was not possible to use only HTML because it can neither make the web pages dynamic nor manage a database. There exist many tools for making web pages such as *Adobe GoLive, Microsoft FrontPage, FileMaker's Home Page* and *Macromedia Dreamweaver*. Although they can help build nice-looking and in some regards dynamic web pages by creating HTML script, none of these tools have been used for this project. Why? One reason is that they do not support an interaction with a database so it was necessary to use a more powerful programming language that can program other functions than just the format of the web page. Another reason why no tools have been used is that to get the best understanding and control over the code it is better if the programmer writes the code herself. Often the amount of code is much higher if a tool has generated the predefined code than if a good programmer has written a "tailored" code; as a result of this the individual tailored code/program is often more efficient.

There are two different ways of handling the web pages; either the web page code (text file) can be executed on the client's web browser or it can be executed on the server. This depends in fact on the script language and the terms "client"- or "server-based" script are used to indicate where the code is executed, for example the languages JavaScript and VBScript are client based while ASP and PHP are server-based.

It was decided at an early stage that a server-based language had to be used in the *Web Based Booking System* since a pure client script has no possibility to connect to a database. A server-based language also gives higher security because the code is hidden as much as possible from the user, since it is executed only on the server and not on the client. The question was which server-based language to use. There were several languages to choose from such as PERL, Java, C++ and PHP. In addition there is ASP, a well-known technique when it comes to web programming, which is not a language but a technique for server-based script languages. It is a mixture of a program language and HTML and has the purpose of making web pages dynamic.

When choosing a language one has to not only consider the functionality, security, and performance of the system but the software environment as well. A certain language can work better on a certain operational system and with a certain server and database than with others. It comes down to selecting the best combination of the operating system, server, program language and database to use for the system. One of the two most used combinations today is ASP [5] on the Microsoft NT-based web server IIS (Internet Information Server) and a database such as Microsoft Access; the other is the LAMP combination (the acronym for Linux, Apache, MySQL, PHP). The new ASP.NET (ASP's successor), which Microsoft has developed as the centrepiece of its dot-NET initiative, was also among the alternatives.

By analysing the languages and the requirements of web sites it was judged that one of the languages PHP, ASP or ASP.NET could be used with satisfaction. However, sites that are expected to support very complex and server-taxing applications might perform better with, for example, Java-based middleware instead of either PHP or ASP [2].

4.1.1 ASP vs. PHP vs. ASP.NET

PHP was originally an acronym for Personal Home Page tools but today stands for PHP: Hypertext Preprocessor. The language was initially developed by Rasmus Lerdorf in 1995 and is a so called Open Source code, which means that the language is under continuous development. Propositions of improvement and additions are received by the company Zend which has responsibility for PHP's development. In this way everyone has the possibility to influence and improve the language. In order to execute a web page that includes PHP it requires a server program that can be downloaded from Internet for free, which makes it available for everyone.

PHP works with a multitude of different Web servers (including the top three: Apache, IIS and IPlanet) and operating systems (including Unix, Linux, NT, Windows 2000 and Mac OS X). On open-source-based Apache, PHP can either run as a CGI or it can be compiled into Apache for fast, native execution.

PHP was specifically created for use on the Web. This does not necessarily mean that PHP lets one do things that are not possible in, for example, VBScript, but PHP is designed to make development for the Web as easy as possible.

In addition, PHP enables developers to create object-oriented code and, by modifying the source files that other programmers have already written, to create highly tailored modules without unnecessary work.

Active Server Pages is the meaning of ASP which, as mentioned earlier, is not a programming language but a technical framework for web applications. It is a commercial technology, supported by Microsoft. To work with ASP one has to use different tools that already exist in the Windows developing environment; the most usual ones are the script languages VBScript and Jscript. ASP runs exclusively on the Microsoft IIS server and since it was not created specifically with the Web in mind certain tasks require more steps than PHP [2]. ASP is easy to learn and easy to use. For large web systems where ten or hundreds of thousands of people simultaneously can use the system, the performance of the ASP technique is not sufficient [4].

ASP.NET is not just a dressed-up version of ASP; reportedly it is a complete rewrite of the technology, the result of four years' work by a separate team of programmers. Here are four of its main goals [2]:

- Make it simpler to create dynamic Web pages.
- Increase the speed of generating dynamic Web pages.
- Increase the efficiency for generating exchangeable and connectable Web services, which are the core of Microsoft's .NET initiative.
- Make it language-agnostic.

4.1.2 Comparing PHP, ASP and ASP.NET

Table 1. Comparing PHP, ASP and ASP.NET. The table is modified from the article in SeyboldReports [2].

	PHP	**ASP**	**ASP.NET**
Platform and Servers	Supports multiple platforms and servers.	Limited to IIS and Windows 2000/XP.	Same as ASP.
Open source/ commercial availability	Open source + Zend commercial products and support.	Commercial products from Microsoft and support + third-party solutions.	Same as ASP.
External modules	Many external modules available; most are open-source and free.	Fewer external modules available; included with ISS or through third parties.	Same as ASP, but also server control and XML Web services to reuse code or Web applications.
Type of languages	Scripting languages made for the Web, with elements of C, Java, and Perl.	Easier scripting language, based on VisualBasic.	Supports multiple languages. Has two new object-oriented languages, C# made for the Web and VisualBasic.NET.
Reusability	Reusability through COM, Java classes, PHP classes.	Reusability through COM.	Same, plus server controls and support of other Classes.
XML and SOAP support	XML parsing and SOAP creation.	XML parsing and SOAP creation.	More direct and easy to create XML and SOAP.
Scripting/ object-oriented	Interpreted scripts with compiled classes, semi-object-oriented.	Interpreted scripts with compiled classes.	Object-oriented and compiled code while maintaining scripting; output caching.

4.1.3 Conclusions

ASP has been the dominant web server "language" during a long period but its dominance is now less obvious. In fact, Apache is by far the favourite server, with a market share of 67 percent (in August 2002) of the active servers, versus 25 percent for IIS, according to Netcraft [16], a British research company that scans the Internet to decide the use of Web server technologies. Because ASP is part of the standard IIS installation, we can infer that these sites at least had access to ASP. According to SecuritySpace, a division of Internet security consulting company E-Soft Inc., PHP is now installed on 38 percent (August 2002) of all active Apache servers, down from 43 percent a year ago. ASP.NET is too new to be in a survey.

For this particular Web based system any of PHP, ASP, and ASP.NET would work and normally in such a case the language or technique is often chosen depending on the company's or programmer's, past experiences. However, since I had no experience of any of these languages and techniques I chose the language which I was most interested in learning. After the analyses, PHP was chosen as the language to be used. It would be used on the web server IIS, since that was one of the requirements. There were several reasons why I chose PHP:

- The fact that PHP works on many servers, OS and databases makes it more flexible, thus making it simpler for the customer to eventually change the environment requirements of the product, if so desired.
- I found the syntax very similar to Java, which is the language I feel most comfortable with.
- The web site www.php.net, with its introductory tutorial, is a great site for learning PHP. In addition, it includes letters with answers from PHP users all over the world, since everyone can send questions and have them answered within a short period of time (sometimes within a few minutes).
- According to a Master's thesis report [1] PHP is advancing much more than ASP and many programmers change from ASP to PHP because they find that PHP has "more possibilities" for making dynamic web sites.

4.2 Choosing the database model

For this project there had to be a database involved in which to store the member and booking data. However, it was not decided from the beginning which database and Database Management Systems (DBMS) were best to use. As history shows, conceptually different DBMSs have been developed but they all were made to handle large amounts of data, and still have the following characteristics:

- Permanent storing of data – data are stored and can be reachable for the program even after the computer has been closed down and restarted.
- Handling storages – when the data is too large to be stored on the primary memory it has to be removed to the secondary memory.
- Dividing data – to make the database efficient the same data must be divided among different clients. User transactions are one way to realize this.
- Data integrity – to make sure the data that is being manipulated in the system is of correct type.
- Data trust – the data that is given out to the user must be correct.

- Data safety – to provide a certain safety for the database there must be some type of rules for administration, in other words, not all users are allowed to manipulate the data.
- Query facility – to enable users retrieving usable data there should be some kind of query language or interface that the users can use to find the data in the simplest possible way.

The database concepts that have been analysed are three of the most widely used ones, the relational, the object-oriented, and the object-relational database.

4.2.1 Relational databases

The Relational Model of Data was first presented to the world in 1969 by Edgar F. Codd. The model offered at that time a conceptually different approach to data storage. In the relational database, all data is represented as simple tabular data structures (relations), which may be accessed using a high-level non-procedural language. This language is used to gain access to the relations and the desired set of data and the programmer does not have to write algorithms for navigation. By using this approach the physical implementation of the database is hidden, thus the programmer does not have to know the physical implementation to be able to access the data. In 1974 Chamberlain and others at IBM proposed such a high-level non-procedural language—SEQUEL, which later, due to legal problems, was renamed SQL in 1990. In 1986 the ANSI X3H2 committee accepted SQL (SEQUEL) as an ANSI standard, and soon SQL's biggest competitor, QUEL, vanished.

4.2.2 Relational structure

A relational database is a finite set of relations (which includes sets of tuples). A relation includes as a matter of fact two things in one: a set of relation schemas called a database schema, and a set of corresponding relation instances called a database instance. Figure 4 shows one possible fragment of a database schema while Figure 5 gives examples of instances corresponding to these relation schemas. One can see that each relation satisfies the type constraints specified by the corresponding schema. The constraints are necessary for the structure of a relational database.

STUDENT (Id : INTEGER, Name : STRING, Address : STRING, Status: STRING)

PROFFESSOR (Id : INTEGER, Name : STRING, DeptId : DEPTS)

COURSE (DeptId : DEPTS, CrsCode : COURSES, CrsName : STRING, Descr : STRING)

TRANSCRIPT (StudtId : INTEGER, CrsCode : COURSES, Semester : SEMESTER, Grade : GRADES)

Figure 4. Relational schema.

PROFESSOR	Id	Name	DeptId		
	10203	John Smyth	CS		
	36275	Mary Doe	MGT		
	94767	Ann White	EE		
	67843	David Jones	MAT		
TRANSCSIPT	StudId	CrsCode	Semester	Grade	
	1111	MGT123	F1994	A	
	1111	EE101	S1991	B	
	3333	CS201	F1993	A	
	4444	MAT110	S1994	C	

Figure 5. Corresponding Instances.

4.2.3 Constraints

The constraints are needed to provide the data integrity and data trust. They are also needed to capture the notion that elements of different sets are related to each other. A constraint is a statement about all legal instances of a database. That is, to be qualified as a legal instance, a set of relations must satisfy all the constraints associated with the database.

Type constraints are needed to define the structures of the schemas and the values of the attributes. Each column in a relation instance must correspond to an attribute in the relation, and the column names must be the same as the names of the corresponding attributes. Each attribute value must be one of the primitive types, such as strings, integers, characters etc.

A *key constraint* means that the value of the Id attribute in all rows of an instance must be unique. A key can consist of one or several attributes. In the example of the student record database, it is reasonable to assume that a student can get one (final) grade for any course in any given semester; therefore the combination {StudId, CrsCode, Semester} is the key in the Transcript relation. A relation can have more that one key but one of them is designated as the primary key.

To create relations between the tables the foreign keys are required. A *foreign key* means that the value of the attribute Id in one relation is the same as the value of the Id attribute in another relation, see Figure 6.

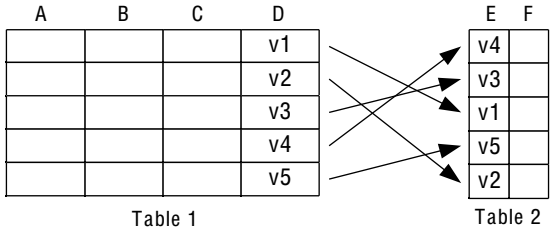


Figure 6. Foreign keys.

4.2.4 Query languages

There are several different query languages that can be used to retrieve data from a relational database, for example Relational Calculus, Tuple Relational Calculus (TRC), Domain Relational Calculus (DMC), and Structured

Query Language (SQL). They are different but they are all built on the relational algebra. The language that is most widely used is the SQL [9]. The reason why it has become so popular is probably because of the exclusive use of normal English words in the syntaxes instead of mathematical terms. Therefore it is easy to learn and handle. The latest version, SQL:99 (also called SQL3), includes some constraint extensions which are explained in section 4.2.8 of this report.

It is impossible to write complete applications entirely in SQL, so database applications are typically written in a host language and they access databases by executing SQL queries embedded in a host program. One problem with this approach is that SQL is set oriented, meaning that its queries return sets of tuples. In contrast, C, Java, and other host languages do not understand relations and do not support high-level operations on them. The possibility of eliminating this mismatch of types was one reason for the development of object databases in the early days.

4.2.5 Object-Oriented databases

As the relational database was being improved and established on the market the new ideas of object-oriented program languages were introduced to the computer program world. At the end of the1970's researchers started to think of the possibilities of creating a database based on the object oriented language technique. At the end of the 1980's the first object-oriented databases entered the market, such as ObjectDesign, Versant, O2, and Objectivity. It was rather difficult though to gain market share since the relational database was well established by then.

4.2.6 Object-oriented structure

An Object-oriented Database Manger (OODBM) has the functionality and properties of an object language and a database. To create such a system means to integrate a database and object-oriented technology. The purpose is to improve databases with object-oriented concepts, or to improve the object-oriented language with permanent storing of data.

An object database consists of classes, which are sets of objects. Each object has a unique key, an object Id and a value, (OId, Val). OIds are used for identifying and finding the objects when they are stored on the secondary memory on the computer where the database server is installed. When the object is requested it is moved to the primary memory and the OId is changed to a pointer that shows where the object is stored in the primary memory. One way to decide if the object is in the primary or secondary memory is to create a directed graph in the virtual memory. The objects are represented as nodes and the references are represented as the directed edges. Each edge has one bit to indicate if the edge is a pointer or an OId.

The benefit of having pointers to the object's positions is that one gets direct access to the objects. Objects can be organized into an inheritance hierarchy, which allows objects of a lower type to inherit the attributes and methods from objects of a higher type. The values of the object can be of complex types such as sets, tuples, and references to other objects. This is an example [9] of an object that describes a person:

```
(#32,      [    SSN: 111-22-3333,

           Name: Joe Public,

           PhoneN: {“516-123-4567”, “516-345-6789”},

           Child: {#445, #47} ]   )
```

The symbol #32 is the OId of the object. The value in the example is a tuple with attributes and attribute values.

4.2.7 OQL (Object Query Language)

OQL is used for retrieving data from the object-oriented database. It is not used for updating or creating new objects in the database and therefore has no support for functions such as views, constraints, or stored procedures. The syntaxes for OQL are similar to those of the SQL but some supplements have been made as for example object identifying, complex objects and inheritance. This is an example of how an OQL question may look:

```
Select c.address

from Persons p,

p.children c

where p.address.street = “Main Street” and

count(p.children) > = 2 and

c.address.city ! = p.address.city
```

The question is taken from ODMG standard (ODMG 2.0 ISBN 1-55860-463-4) and represents Figure 7. The query inspects all children of all “Persons” to find people who live on “Main Street” with two or more children. It returns only those addresses of children who do not live in the same city as their parents. It navigates from the Person class using the child reference to another instance of the Person class and then to the Address and City classes.

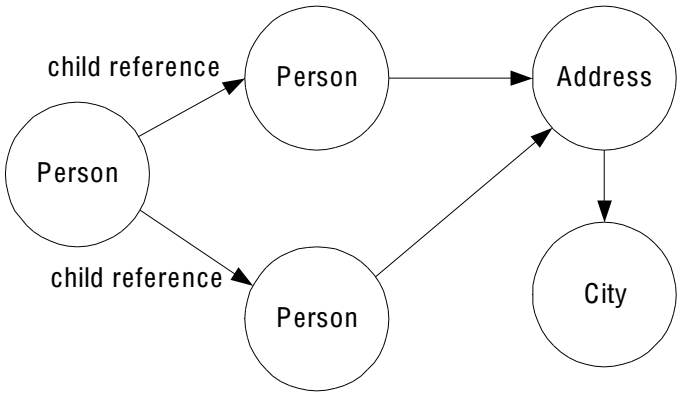


Figure 7. Navigating a data structure with OQL.

4.2.8 Object-Relational

The object-relational DBMS (ORDBMS) is the newest commercial breed of DBMSs, which embrace some object-oriented features and encapsulate these features into an RDBMS, creating an ORDBMS. ORDBMSs are mainly based upon the criteria defined by Dr. Michael Stonebraker in 1990 [14]. He suggested extending the capabilities of an RDBMS to include support for richer object structures and rules and still be open to other systems. His suggestions were accomplished through thirteen propositions of requirements to ORDBMSs as extensions to RDBMS.

4.2.9 Object-relational structure

The basic idea behind object-relational databases is that relations are still sets of tuples but tuple components can be objects. There are earlier products that are built on the object-relational model, but soon after its release SQL3 became the most known version. An SQL3 database consists of:

- A set of relations, which can be viewed as classes.
- Each relation is either a set of tuples or a set of objects. They can be viewed as instances of the class that represents the relation.
- An SQL3 object is a pair of the form (OId, Val), where Val is a tuple value. An SQL3 tuple value has attributes and attribute values, which can be primitive values, sets of tuples, and references to other objects.

SQL3 allows abstract data types and calls them user-defined types (or UDT). The UDTs can have methods defined to the type and inherit attributes and methods from other relations. The methods are often written in some other language, for example C, and when they are defined the link to its executable code must be told. It is only through the UDTs the SQL3 objects can be created.

4.2.10 SQL3

SQL3 language syntax looks like the previous SQL versions. The difference is the object-relational extensions. The syntax for retrieving data from objects does look similar to the OQL. For example,

```
SELECT T.Student.Name, T.Grade

FROM TRANSCRIPT T

WHERE T.Student.Address.Street = 'Hollow Rd.'
```

where *Student* is a UDT.

4.2.11 Conclusions of the choice of database

The most significant difference between object-oriented and relational databases is that the relational database is based on the mathematical relational theory while the object-oriented database is based on the object-oriented techniques. Some advantages and disadvantages of relational and object databases are:

- Relational table structures are easy to understand; to understand an ODBMS a user should have knowledge of an object-oriented programming language.

- Complex data relationships are difficult to model in relational and object-relational databases while an object-oriented database can handle arbitrary complexity.

- The RDBMS model provides independence of the data from applications that use the data. This means that each new application does not require any modification of the database.

- It is easy to extend the ODBMS's functionality by defining new classes, including structures, operations, and relationships.

- RDBMS databases have been under development and used much longer than object-relational databases. In addition, SQL is the most universal

database language, while different OODBMSs have different query languages.

- The relations in an object-oriented database consist of pointers, which make it very fast to find objects that are related to another object. On the other hand, the function of searching for sets is not well evolved in the object-oriented database, which it is in the relational database.

One cannot say if one DBMS is faster or slower than the other. It depends on what the database shall be used for. If the database structure is complex and one shall navigate between the objects in order to, for example, find a telephone number which is stored in another object/table, then the OODBMS is the fastest according to Dr. Michael Stonebraker. The question comes down to whether or not the system is complex. The *Web Based Booking System* has a rather simple computer system, and the tables in the database are not going to store more complex data types than INTEGER, CHAR and BOOLEAN, therefore there appeared to be no need to choose an object-oriented or object-relational database. The choice of programming language is also an important factor when it comes to choosing a database. In this case the chosen programming language was PHP, which according to Viktor Jonsson [5] is probably the language that has better support for databases than any other language. PHP supports over 20 different databases; among them are MySQL, Oracle, Sybase, PostgreSQL, SQL server and ODBC. Because the WeBaBoS was not going to be designed based on an object-oriented technique, even though it was possible with PHP, the final choice of database was MySQL. MySQL is a relational database and is today the most popular open source database server in the world with more than four million installations powering websites, data warehouses, business applications, logging systems, etc.

4.3 User transactions

A transaction is a program that can perform the following functions:

1. It can update a database to represent a real-world event.

2. It can guarantee that one or more real-world events have occurred. The transaction will commit if and only if each step in the transaction was completely successful. This all-or-nothing commitment is also called global atomicity.

3. It can return data from the database.

In large systems with many thousands of users at the same time, also known as Transaction Processing Heavy systems, it can be necessary to use some kind of transaction manager. A transaction manager is often an off-the-shelf middleware component that can be used to build TP-Heavy systems. The transaction manager's main purpose is to guarantee global atomicity [9]. In smaller systems that use only one or several databases from the same vendor, like the WeBaBoS system, which for the moment only uses one database, the algorithm for implementing global atomicity is usually integrated into the individual servers provided by the vendors.

In the WeBaBoS system the number of users is set to be maximum of 100 and the queries to the database are single queries and not sequences of queries that are depended of each other's answers. This means that it is a light Transaction Processing system, or TP-Lite. The main transaction problem of the WeBaBos would be when two users or more try to book the same time slot of the same facility at the same time. One way of trying to avoid this phenomenon is to not allow duplicated primary keys in the database structure. When one of the users have gotten the key and the apartment number inserted into the facility table the next time the same key is tried to be inserted the SQL query will return false, which means the key did already exist and could therefore not be inserted again. To make sure a booking did succeed the function which executes the SQL query is always being checked whether the query failed or succeeded, and in case of a failure, an error message is sent out to the user. This control is done inside the application, but can the problem be solved with help of the database manager?

In MySQL the transaction manager is called InnoDB. The InnoDB's goal is to combine the best properties of a multi-versioning database with traditional two-phase locking [19]. A consistent read, see Figure 8, means that InnoDB uses its multi-versioning to present to a query a snapshot of the database at a point in time, which contains the results of all transactions that committed at a prior time. A commit means that the changes made in the current transaction are made permanent and become visible to other users. The normal query, SELECT, does not set any locks on the tables it accesses, and therefore other users are free to modify those tables at the same time as a consistent read is being performed.

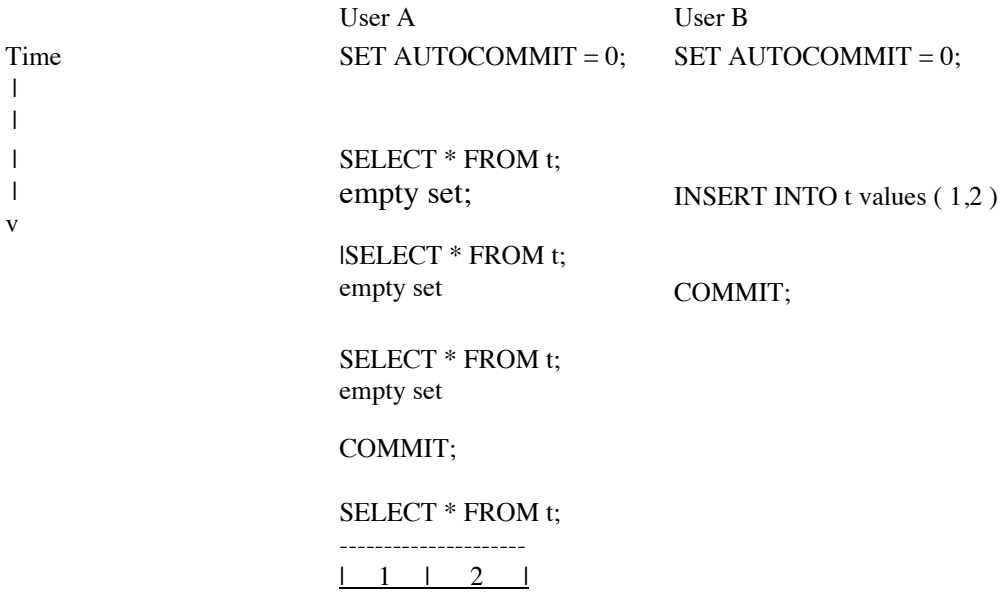


Figure 8. Consistent read.

In some circumstances the consistent read is not convenient, for example in the WeBaBoS case when two users might book at the same time. Suppose there is a consistent read of the table BASTU and they indeed see that the time slot is still available. If so, can one safely add the new booking? No, because it may happen that meanwhile some other user has inserted that time slot into the table BASTU. One solution of this problem is to perform the SELECT in a locking mode, like this:

SELECT * FROM BASTU WHERE time_slot='13-14' FOR UPDATE;

This solution is called the next-key locking, which means that if the data is read in this mode then another user cannot insert new index record R immediately before R in the index order. In addition, a locking read always

gives the “freshest” state of the database because if the latest data belongs to an uncommitted transaction of another user, the read will wait until that transaction commits.

Deadlock is a classic problem in transactional databases. A deadlock occurs, generally speaking, when there is a wait loop caused by a sequence of transactions, $T_1, T_2,\ldots, T_n$, in which each transaction T_i is waiting to access an item locked by T_{i+1}, and T_n is waiting to access an item locked by T_1. According to MySQL’s homepage [19] a deadlock can occur even in the case of transactions, which just insert or delete a single row. That is because these operations implicitly set locks on the index records of the row inserted/deleted, although these locks are not next-key locks and do not prevent other users from inserting or deleting. At MySQL’s homepage there are also some tips of how to reduce the number of deadlocks and one of them is to use less explicit locking since some operations have implicit locks. If one can afford a SELECT to return data from an old snapshot, it is better not to add the clause FOR UPDATE to it.

The advice to not use a next-key lock was taken because solving the problem by checking whether the INSERT query failed or not, is a solution that always works without increasing the risk of deadlock.

4.4 State chart of WeBaBoS

Figure 9 represents the state chart of the system design that shows how the web pages interact.

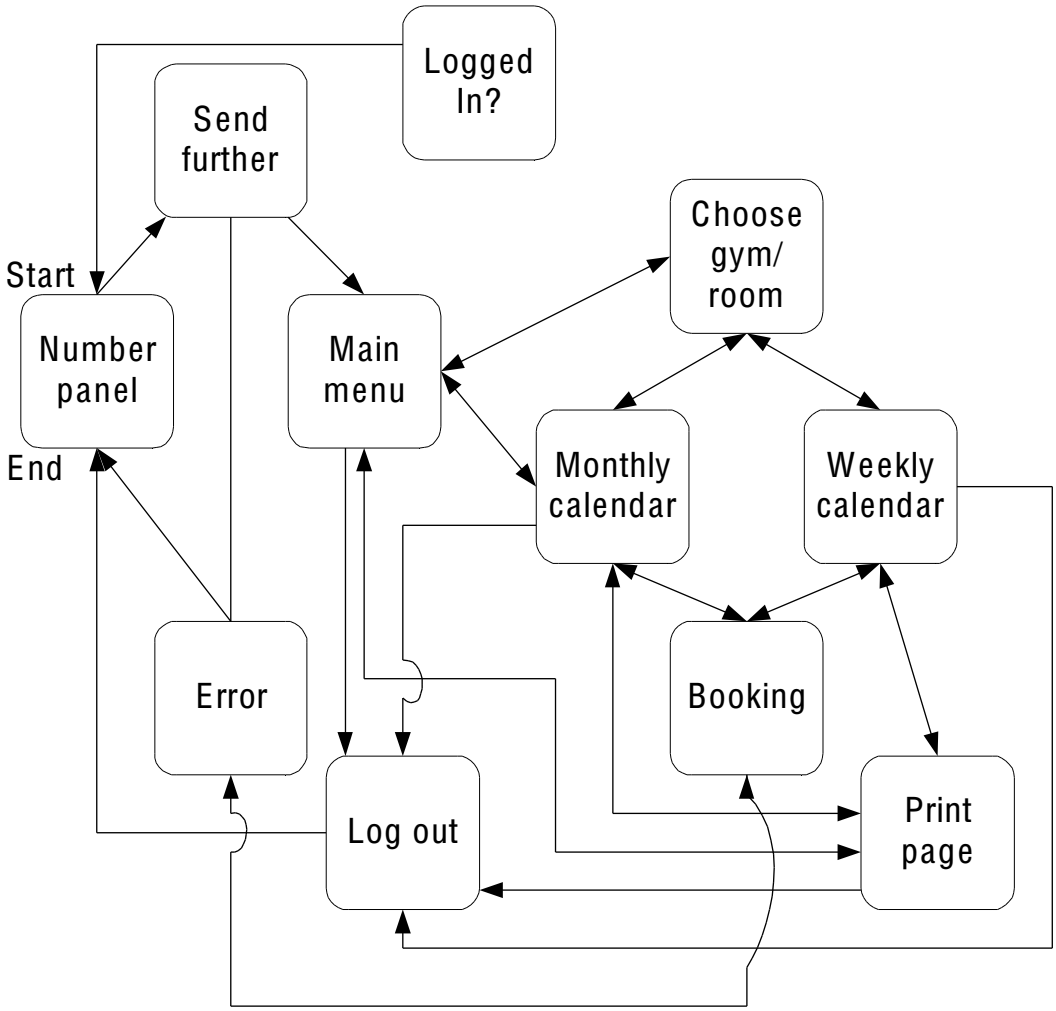


Figure 9. State chart of the web pages. Each square in the chart represents a web page. The arrows indicate to where each page may transfer depending on the user's action.

5 DD phase of the WeBaBoS project

During the DD phase the code was written, tested, and documented and the system got its final design. Even though the system design had been implemented beforehand in the ADD, the design changed rather much during the development of the software. For example more tables in the database had to be added. The reason for this was that a better understanding of how web pages are built and how they are interacting with each other led to a different design, and the ADD had to be updated. The outputs of this phase were, according to the model, supposed to be the code, the SUM, and the DDD, but out came only the SUM (Appendix C) and the program code. The DDD was in this case, judged by the project manager, to be superfluous since the system was not large and complex, therefore the code together with the SUM was enough documentation.

5.1 Software environment

To simulate the internal LAN of the system a small LAN was set up consisting of a Windows Internet server IIS and two clients, one of them with a touch screen. Apart from the small LAN there was an external client with Internet access representing an arbitrary client, for example a computer in a user's office. After this environment was set up the PHP support and the MySQL database were downloaded for free into the server. The database included two GUIs, one for administrating the database, for example making changes in the "my.ini"-file where configuration variables are stored, and one that serves as a client that can manipulate the database by executing SQL queries. They were both very nicely designed with colourful texts and simple to use, even for a beginner.

5.2 Difficulties during software development

The PHP language was easy to learn but since it is a server based language its main purpose is to work as the middleware between the server and the database. A beginner in Internet programming, as I was, could imagine that PHP would be the main language in the sense it would have the main control of the system. Unfortunately it is very dependent on the HTML. It cannot, for example, generate new web pages, except by regenerate the current page, without going via the "meta" tag in HTML. The "meta" tag can also "refresh" a page but not simultaneously using the predefined "goback" function which causes confusion when two users try to book the same time slot. This problem is described in more detail in section 6.2.

HTML is a formatting script with few variations. For instance the buttons are predefined without any possibilities to change either the size or the

colour of them. Because of this limitation all the buttons in the GUIs are painted images to provide the required sizes. In addition areas on a page, for example squares in a table, cannot be linked unless there is a text in it, which is linked. This forced the programmer of WeBaBoS to put the unplanned text “FRI” (free) inside an available time slot in the calendar table.

6 TR phase of the WeBaBoS project

The system has been tested from many different aspects. The tests are documented in the STD (Appendix D), which has been reviewed and approved by the customer.

6.1 Requirement tests

First, the positive scenarios of the requirements from the URD were tested to verify if all the requirements were implemented and functional; then the negative scenarios were checked if the correct error message was generated by the corresponding error handling. The only requirement that has not been implemented is cipher coding over the network. To be able to use the already existing cipher coding program in Windows IIS, an expensive certificate was required and therefore Robert Frick, the customer, decided to wait to order the certificate.

6.2 Simultaneously booking test

When several users try to book the same time slot of the same facility at exactly the same time has been tried to simulate. This is quite difficult to simulate since the database server takes care of transactions between the several users. When a user tries to book a time slot, which seems available but is meanwhile booked by another user she gets an error message. The message informs the user of the double booking and gives the option to go back again to the previous page. When the previous page is regenerated it is refreshed and the user can now see the apartment number of the other user who booked the time slot. This phenomenon can be annoying and confusing for the user but it will probably not occur often.

6.3 Server and database crash tests

Server crash and database crash have been simulated. By turning off the server or the database in the middle of using the system and afterward analyzing the effects, one can try to predict the consequences of a system crash. When the server was turned off while the logged-in user tried to enter a new site, the result was, after a bit of wait, an Internet Explorer error page saying; “the page cannot be displayed”. However, the bookings that were done before the “server crash” were safely stored in the database. When the database was shut down in the middle of a user session the user was instructed to contact the system administrator because of the disability to connect with the database. Also in this case the bookings that had been done before the shutdown were preserved in the database.

6.4 Web browser tests

The web browser plays an important role for the system's design and performance. The system was designed for the Internet Explorer but has also been tested on Netscape and Opera. When the system was run in these two browsers, the calendars' sizes were different because of different resolution standards. The colours turned out differently and some of them were not painted at all. Apparently some predefined colours, such as "light blue" and "dark blue", in Internet Explorer are not defined in Netscape or Opera and were therefore ignored. That the design appears differently in different browsers is not a problem but unfortunately neither Netscape nor Opera accepts JavaScript, which means the performance is affected. For example, the printing function does not work and a predefined "goback" function is interpreted differently so the user cannot get back to the previous page from the error message site once she got there. These problems can be adjusted, but so far Internet Explorer is the only browser on which the system works without complications.

6.5 GUI design tests

The system will be used by a large group of different people; presumably most of them will have little or no computer experience. Therefore one of the requirements was that the GUIs had to be designed with special attention to the capability of the users, and ease of use of the touch screen. The only way to verify if the GUIs fulfill the requirement was to let different people with varying computer knowledge try using the system. The "guinea pigs" managed to adopt the system quickly and had no direct complaints. As would be expected, people had different opinions of the colours and the design, but importantly they all could manage the system without difficulties.

7 Results

Since the system is not in use the WeBaBoS project has not yet come to the OM phase. Therefore it is time to present the results of the different parts of the project.

7.1 The Waterfall model

The software life cycle model, the Waterfall model, has been an appropriate model for this project. The model has been followed phase by phase but not all five documents have been written. For example the SRD that was merged into the ADD and during the last phase, the Detailed Design phase, the output should be a DDD and SUM but in this project there were only the code with documentation and the SUM that were produced. The reason why a DDD was not produced is the simple fact that it was unnecessary, as judged by the project manager, because the substance of a DDD would be too thin. Even though the Waterfall model has worked well it would have been possible to use an even simpler model for this project.

7.2 PHP and MySQL

The PHP language has worked well on the Windows IIS but would probably work even better on an Apache server since it is developed for this server. The PHP was very easy to learn and there are a few well-written books for PHP beginners [5] [7]. Although none of the books about PHP, that I could find, included complicated technical details. To find solutions of technical PHP problems the web page www.php.net had always the answers even though it took some hours to search through all the related questions with answers that are published on the site before the solutions could be found. The PHP has also worked well with the MySQL database. There were simple functions for connecting with the database, integrating the SQL queries, and to handle the result sets.

7.3 The WeBaBoS database structure

The database has been the most essential part in the WeBaBoS. To provide safe management of information, general code, and simpler administration of the system, all the significant data have been stored in the database. The database consists of these tables:

member (cardnumber char(20) default '' NOT NULL,
name char(40) default '' NOT NULL,
member_number char(20) default '' NOT NULL,
code char(10) default '' NOT NULL,
ap_number char(10) default '' NOT NULL,
sex char(8) default '' NOT NULL,
attempts char(20) default '' NOT NULL,
timestamp char(20) default '' NOT NULL,
extra_1 char(20) default '', extra_2 char(20) default '',
PRIMARY KEY (cardnumber))

facility_data (fac_name char(20) default '' NOT NULL,
fac_no char(20) default '' NOT NULL,
max_slots char(20) default '' NOT NULL,
max_days char(20) default '' NOT NULL,
extra_1 char(20) default '',
extra_2 char(20) default '',
PRIMARY KEY (fac_name))

Tvättstuga_1 (date char(20) default '' NOT NULL,
ti_slot char(10) default '' NOT NULL,
ap_number char(10) default '' NOT NULL,
PRIMARY KEY (date,ti_slot))

general_data (max_inlogged char(10) default '100' NOT NULL,
automatic_logout char(10) default '3' NOT NULL,
max_attempts char(10) default '3' NOT NULL,
lock_time char(10) default '60' NOT NULL,
extra_1 char(10) default '',
extra_2 char(10) default '',
PRIMARY KEY (max_inlogged))

logged_in (cardnumber char(20) default '' NOT NULL,
ti_stamp char(12) default '' NOT NULL,
PRIMARY KEY (cardnumber))

The *member* table stores information about the residents in the building. Everyone shall have, according to Robert Frick who is the orderer of the system, a personal card, member number, and code with which they can log in to the system. The “attempt” variable represents the number of times a person may try to log in with an incorrect code before the card number will be locked. The timestamp in the table is the exact time when the lock was activated.

The *facility_data* has information about the facility's constraints, such as how many time slots, "max_slots", a user can book during a period, and how many days, counted from the current day until the variable max_days, the period include for that particular facility. The calendar in the system depends on the maximum number of days the facility can be booked in advance. Only the days within the period are linked to the booking application and can therefore be booked.

The *Tvättstuga_1* table is one of the facilities that can be booked. In this system there are six laundry rooms but each room has its own table in the database, just like the other facilities. Each of these facilities includes a date and a time slot, which together make the primary key. This is a reasonable solution since only one family at a time can use the facility at that particular day and time.

The *general_data* table stores data that has to do with the special requirements. The first column represents the maximum number of logged-in users allowed at the same time. It is by default set to 100 users because that is one of the requirements in the URD. The second column value in the table is the time, measured in minutes, before the automatic logout is activated if no buttons or links have been activated during these minutes. The "lock-time" variable is the time before a lock on a card number is unlocked again. The lock is set by default to be locked during one hour (60 minutes) if the lock has been activated.

To provide security, the system's web pages are inaccessible unless the user logs in. When a log-in is being successfully carried out the user's card number will be stored in the *logged_in* table together with the timestamp (ti_stamp) of when the user logged in. Each time a new web page is entered the logged_in table is checked. If the user's card number is in it and if the timestamp has not exceeded the "automatic_logout" time the user accesses the page, otherwise the log-in page is shown.

7.4 The GUIs design – Sunny Day Scenarios

Figure 10 represents the Log-in page from where the users can log in by first pressing their card against the card reader and then type in the code by pressing the buttons. If the user logs in from an external client the member number and code can be typed in via the keyboard.

Figure 10. Log-in page.

By pressing directly on the facility's name on the main menu page, see Figure 11, the user will come to the facility's booking schedule. If the user chooses to press the "ALLA BOKNINGAR" button (all bookings) every booking (within a month period) that holds the user's apartment number will be listed.

Figure 11. Main menu page.

The user can have a receipt of the bookings printed by pressing the “JA” button (YES), Figure 12.

Sara Hvatz i lägenhet 543 har bokat
Pingislokalen den 11:30:2002 mellan tiden 19_20
Tennisbanan den 12:01:2002 mellan tiden 14_15
Bastun den 12:15:2002 mellan tiden 19_20
Gym_2 den 11:29:2002 mellan tiden 16_17
Tvättstuga_4 den 12:08:2002 mellan tiden 13_16
Vill du ha ett kvitto?
JA
LOGGA UT
HUVUDMENY

Figure 12. Receipt page.

If there are more than one facility of the same type, the user has to choose which one of these he would like to access, see Figure 13.

Figure 13. Different laundry rooms.

The laundry rooms have week schedules (Figure 14), which by default start on the current day. By pressing a green square with the text “FRI” (FREE) the booking will be performed. If the booking succeeded the user’s apartment number on an orange background will replace the green square. The orange colour represents the users own bookings while other users’ bookings are represented with a pink colour and respective apartment number. To change a booking the user just has to press the square with her own apartment number and the square will become available again.

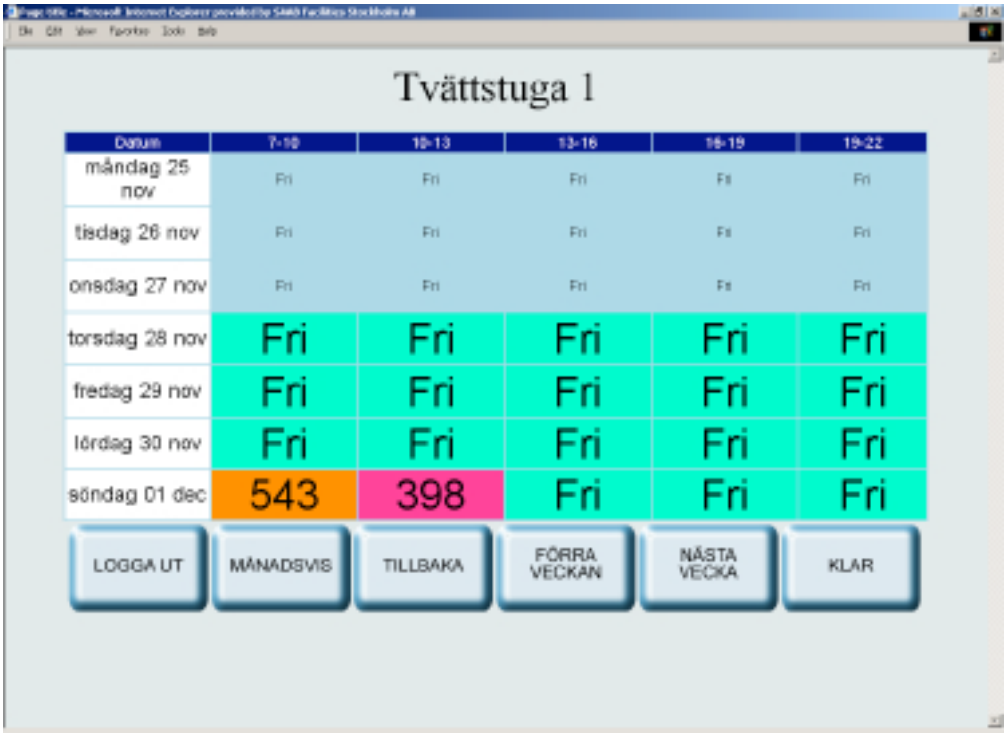


Figure 14. Laundry room’s booking schedule (first week).

The schedule is bounded by the facility's restrictions (Figure 15). On this picture one can see that the time slots on the Sunday of this week cannot be booked since they are not linked, they are simply too far ahead in time. However, the schedule changes every day, which means the time slots were available the day after this picture was taken. The "NÄSTA VECKA" button (next week) shows the next week but the days are not linked.

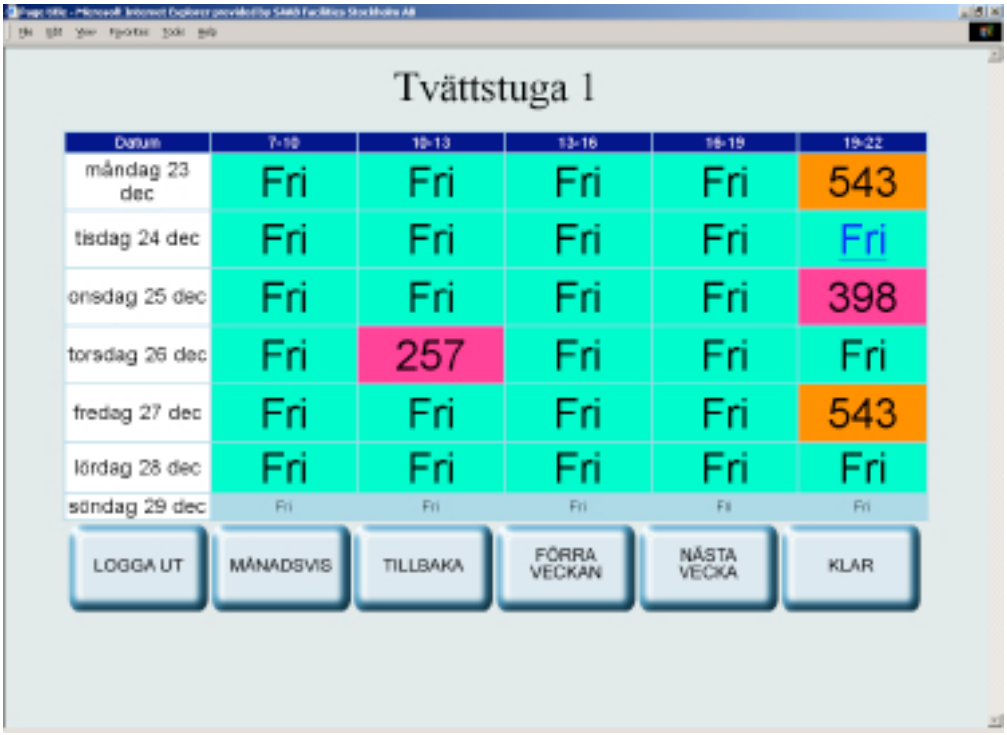


Figure 15. Laundry room's booking schedule (end week).

The “KLAR” button (done) will list all the bookings of all the current facilities. To list all of the current facilities helps the user to see which one of them she has booked if by chance she forgot or did not book herself (Figure 16).

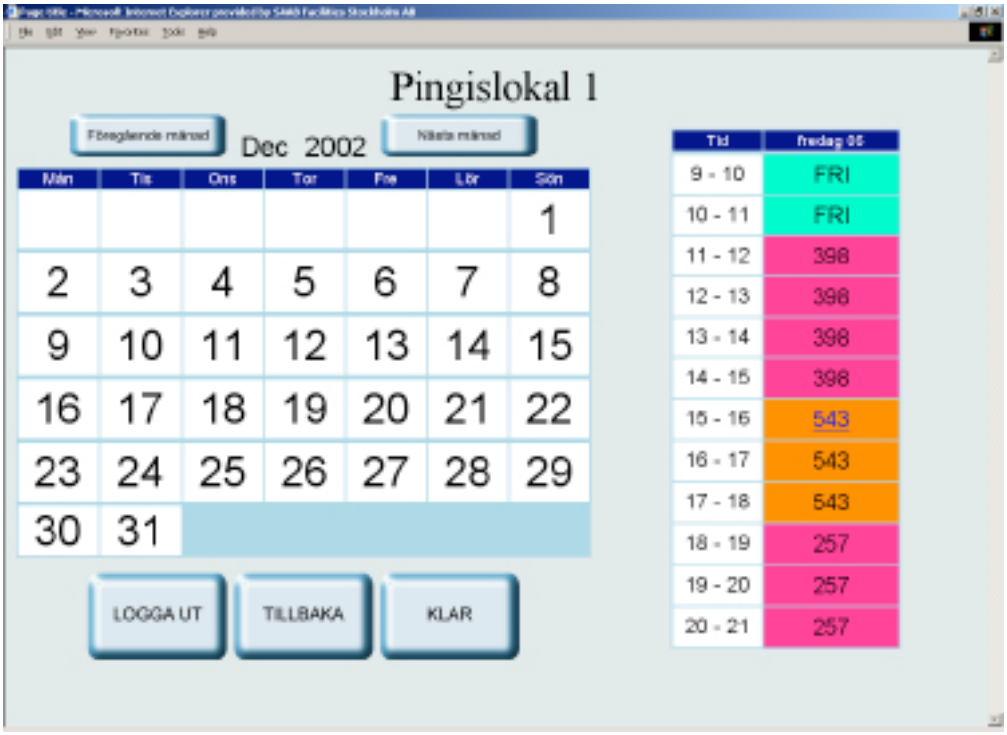


Figure 16. Receipt page for laundry rooms.

The facilities that can be booked for a minimum of one hour have monthly schedules; see Figure 17. It was impossible to squeeze in all the “one-hour” time slots on to one page in a weekly schedule therefore this monthly calendar with one day on the side was made. The principles of booking and changing a booking are the same. The “KLAR” button lists all the user’s time slots of Pingislokal 1.

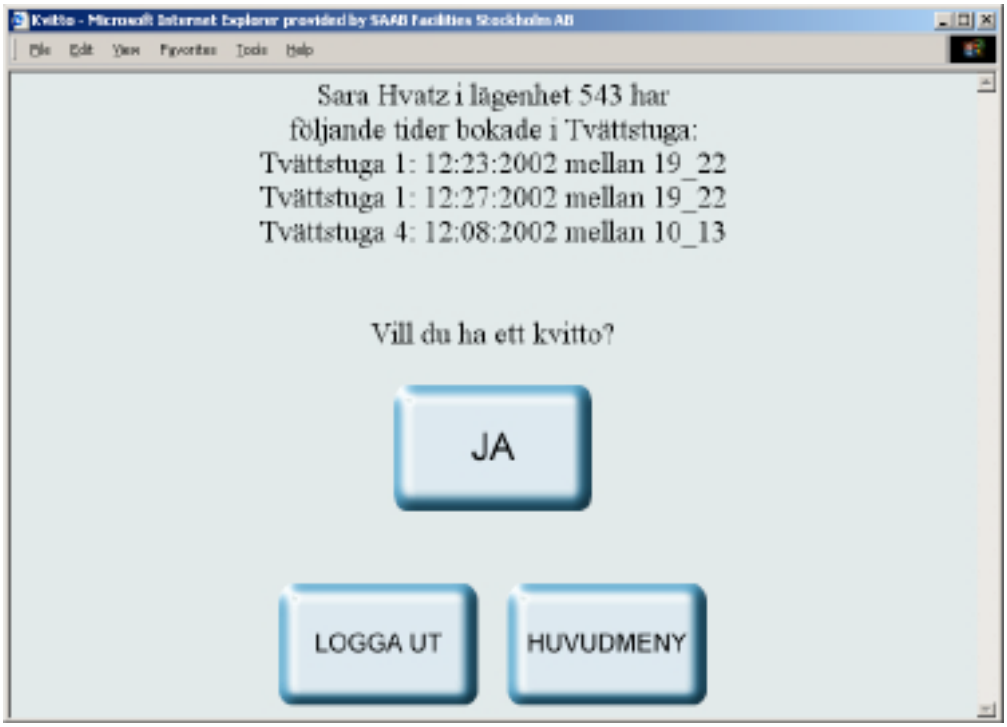

Kvitto - Microsoft Internet Explorer provided by SAAB Facilities Stockholm AB
File Edit View Favorites Tools Help
Sara Hvatz i lägenhet 543 har
följande tider bokade i Tvättstuga:
Tvättstuga 1: 12:23:2002 mellan 19_22
Tvättstuga 1: 12:27:2002 mellan 19_22
Tvättstuga 4: 12:08:2002 mellan 10_13
Vill du ha ett kvitto?
JA
LOGGA UT
HUVUDMENY

Figure 17. Ping-pong room’s booking schedule.

7.5 The GUIs design – Rainy Day Scenarios

These two error messages are examples of how the user is informed of her mistake: the first picture (Figure 18) will occur if the user has typed an incorrect member number or code when he is trying to log in, and the second (Figure 19) occurs when the user presses a square which is already booked by a person, who in this case lives in apartment number 398.

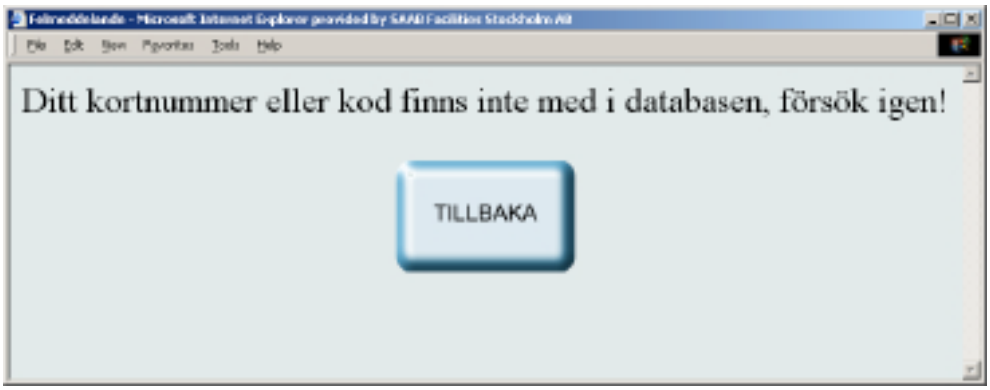


Figure 18. Error message for incorrect card number or code.

Figure 19. Error message for occupied time slot.

7.6 The card reader

To get the card number from the card reader into the “Log-in” web page every internal client has a little C program installed. The program is constantly listening for signals from the card reader and transforms them as keyboard events. When the card number form of the web page is in focus the card number comes into the form as a result of the generated stream of keyboard events.

7.7 The printer

The printer on each internal client activates when the user clicks on the “JA” button from the receipt page.

7.8 Maximum performance

It is often difficult to estimate the maximum performance of computer systems and the WeBaBoS is not an exception. I have tried to do an estimation of how many users can be logged-in at the same time without significantly reducing the system’s performance, but I have not come to a reasonable conclusion. In MySQL’s manual it says it can store several hundreds of thousands of rows in a table but there are no time estimations of how long it may take to manipulate data of that size. Likewise, I have not found any useful information of how powerful Microsoft’s Internet Information Server is.

7.9 The Time plan and Limitations

The time plan, which was set up early in the project, has been followed closely without difficulties. The project’s 20 weeks were spent on the following elements and the time spent on each element is indicated as a percentage:

- Interview with the customer, deciding project model, documenting the requirements into the URD. (10 %)

- Searching, reading, and analyzing information useful for the ADD, writing the ADD. (40 %)

- Developing and documenting the software, testing the software and the system. (30 %)

- Writing this report. (20 %)

There was never any need to limit the project, for example by not implementing some of the customer’s requirements, at least not because of lack of time. One thing that I would have spent more time on if I had had more time for this project was the design of the web pages. They are the only communication between the system and the users and are for that reason very important. The GUIs at the moment are simple and have rather large fields to press on, just like the requirements called for in the URD. But my preferences would be to make them more visually interesting and more esthetically coherent, of course still maintaining simplicity.

8 Future thoughts

One of the requirements was that the software of the system had to be as general as possible. This was because SaabTech Systems has ideas of expanding the system. One of their ideas is that the entrance doors of a building complex shall be controlled by the same card reader system as in the internal clients of the WeBaBoS. Robert Frick has also thought of an extension of this idea, which is to allow the door of, for example, the laundry room to be opened only by the persons who live in the apartment that holds the current booking.

As it becomes more and more common to have Internet access in people's homes and at people's work, I personally believe that in the future the internal clients with the touch screen will be superfluous. There can still be an internal LAN with a client in each apartment. If this is the case, the GUIs would be designed totally different and more personal functions like personal calendars, phone number books, and photo albums could be of current interest. Or a common "neighbour-album" with pictures from the barbeque last week. Some of these functions already exist in the Home Gateway system product from Frontyard AB, which I think is a product of the times that can give inspiration to additional functions for the WeBaBoS.

Computer system solutions for simplifying daily life is so far a rather unexplored market with many directions for future development. I think the WeBaBoS has good chances to become a successful product, provided continued development interest from *SaabTech Systems*, and is so far a good start for serving today's booking requirements of the housing associations.

Definitions, acronyms and abbreviations

ADD Architecture Design Document.

ADT Abstract data type.

BLOB Binary Large Object is a sequence of bits.

CGI Common Gateway Interface is a protocol.

CLOB Character Large Object. An SQL CLOB is a built-in type that stores a Character Large Object as a column value in a row of a database table.

COM Component Object Model is component software architecture from Microsoft, which defines a structure for building program routines (objects) that can be called up and executed in a Windows environment. COM provides the interfaces between objects, and Distributed COM (DCOM) allows them to run remotely.

C++ An Object-oriented version of C.

DDD Detailed Design Document.

GUI Graphical User Interface.

HTML Hyper Text Markup Language is used for designing text and images for the Web.

JAVA An Object-oriented programming language.

LAN Local Area Network.

PERL Practical Extraction and Report Language, a programming language that combines syntax from several UNIX utilities and languages.

PHD Project History Document.

SOAP Simple Object Access Protocol lets one XML Web Service link to another. For instance someone could make a rental-car reservation at an airline Web site without retyping the information that both services require. As long as both

companies support XML and a common SOAP protocol, there is no need to integrate their different database backends.

SRD Software Requirement Document.

SQL Structured Query Language is a language for manipulating a relational database.

STD Software Transfer Document.

SUM Software User Manual.

TCP/IP Transmission Control Protocol/Internet Protocol is a routable protocol, and the IP part of TCP/IP provides this capability. In a routable protocol, all messages contain not only the address of the destination station, but also the address of a destination network. This allows TCP/IP messages to be sent to multiple networks (subnets) within an organization or around the world, hence its use in the worldwide Internet.

URD User Requirement Document.

WeBaBoS Web Based Booking System.

XML Extensible Markup Language, a technique for defining data elements on a Web page and business-to-business documents.

References

[1] Luiga M., Melin K. & Wåhlin A-K. (2002), *PHP kontra ASP*. University of Kalmar: Faculty of Media Science and Journalistic. [In Swedish]

[2] Hartman H. (2001). *Tools for dynamic Web sites: ASP vs. PHP vs. ASP.NET*, SeyboldReports Analyzing Publishing Technologies, Volume 1, no 12, pages 3–8.

[3] Sandberg D. (2000). *ASP och PHP – lika men ack så olika*, Nätverk & Kommunikation, no 7, pages 58–60. [In Swedish]

[4] Karlander L. (2000). *Så långt håller ASP-kosymen*, Datateknik 3.0, no 2, pages 31–33. [In Swedish]

[5] Jonsson V. (2001). *Webbprogrammering med PHP*. Lund: Studentlitteratur. [In Swedish]

[6] Kihlman D. (2001). *Programmering för Internet*. Lund: Studentlitteratur. [In Swedish]

[7] Ek J. & Eriksson U. (2001). *PHP4 programmering*. Stockholm: Pagina förlags AB. [In Swedish]

[8] *ESA software engineering standards (01/02/91),* ESA PSS-05-0 Issue 2, International Number: VII/24.DOC.

[9] Lewis P. M., Bernstein A., Kifer M. (2002). *Databases and Transaction Processing: an application-oriented approach*. USA: Addison-Wesley.

[10] Skoog S., Vasiljevic D. (2001). *Objektorienterade databaser: funktionalitet och systemutveckling*. The Royal School of Technology (Stockholm). [In Swedish]

[11] Asbjørn D. (1998). *The Evolution Of Data Models And Approaches To Persistence In Database Systems*. University of Oslo: Department of Informatics.

[12] McClure S. (1997). *Object Databases vs. Object-Relational Databases*, IDC Bulletin #14821E.

[13] Catell R.G.G. et al. (1997). *The Object Database Standard: ODMG 2.0*. USA: Morgan Kaufmann.

[14] Stonebraker M, Rowe L.A., Lindsay B., Gray J., Carey M., Brodie M. L., Bernstein P. A., Beech D. (1990). *Third-Generation Database System Manifesto*, SIGMOD Record, 19(3), pages 31–44.

[15] www.techweb.com/encyclopedia/ Last visited: 12 Dec. 2002.

[16] www.netcraft.com/survey Last visited: 12 Dec. 2002.

[17] www.securityspace.com Last visited: 12 Dec. 2002.

[18] www.asp.net Last visited: 12 Dec. 2002.

[19] www.mysql.com/doc Last visited: 12 Dec. 2002.

Appendix A

User Requirement Document

Degree project

Web Based Booking System

By Sara Hvatz

Contents

A.1 Introduction 53
A.1.1 Purpose 53
A.1.2 Scope of the software 53
A.1.3 Definitions, acronyms and abbreviations 53
A.1.4 References 53
A.2 General Description 54
A.2.1 General Capabilities 54
A.2.2 General Constraints 54
A.2.3 User Characteristics 55
A.2.4 Assumptions and dependencies 55
A.2.5 Operational environment 55
A.3 Requirements 56

A.1 Introduction

The first part provides the purpose of the document itself, a short description of the product, acronyms, and references. The second part describes the system in general and the required capabilities of the end users. The constraints, assumptions, and dependencies of the system are presented as well as the operational environment. The third and last part is the most detailed, in which the requirements of the product are divided into functional and non-functional requirements.

A.1.1 Purpose

The purpose of this document is to mediate the customer's requirements of the product in a precise and accurate way to eliminate possible misunderstandings or misinterpretations. The content is the base of the whole software design process. The report describes the customer's expectations of the system and if the customer would not be satisfied with the finished product this document can work as a basis to find the reason of the dissatisfaction. It is therefore of great importance that both the customer and the supplier have clear agreement before the project begins.

A.1.2 Scope of the software

The software to be developed will be integrated into the already existing hardware for a booking system.

A.1.3 Definitions, acronyms and abbreviations

LAN – Local Area Network
TCP/IP – Transmission Control Protocol/Internet Protocol
SQL – Structured Query Language
GUI – Graphical User Interface

A.1.4 References

The main source is Robert Frick, who in this case serves as the customer, and with whom I conducted an interview. Another basis is the report of Fredrik Åsell, who developed the hardware for this project. For technical questions I can turn to Mats Tholén and Thomas Ljungvall.

A.2 General Description

What SaabTech Systems would like me to develop is a prototype of a booking system to be used in a civil environment. This system is meant to make daily life a little bit easier. A normal scenario may be a person who would like to know if the laundry room or the tennis court at home is free for use. This remote booking system would make it possible for that person to find out, even without physically going home to check on the paper schedule outside the laundry room. All this person would have to do is to connect to Internet, write the web address, and finally log in to the booking system and reserve an available and preferred time spot. When the system is not used for booking objects the first page will work as a general information source for the members. The first page will be on all the time on the terminals and from which the member must log in to be able to use the booking functions. The system contains four main parts: a server, a relation database, terminals (PCs) each connected to a touch screen, and external terminals. An external terminal might be any terminal connected to the Internet. The terminals which are connected to a touch screen are each placed in a special designed box which are suppose to be situated convenient for the users. One possible area to place the boxes might be, as I presented earlier in the text, in an apartment complex where the residents can book the different objects in the complex.

The main functions for the users are to book an object or change a self-made booking. Naturally the administrator has to be able to chance or correct wrongly booked objects, member data etc.

A.2.1 General Capabilities

- The system's internal terminals shall be connected to a LAN and a server.
- The server shall be connected to a database.
- The software shall be as general as possible to facilitate future function addings for e.g. "open a door" function and a camera connection.
- In order to make the software able to work on most modern operating systems the software, will be written in a modern language, for example, Java.

A.2.2 General Constraints

Here are the constraints of the facilities to will be possible to book:

- The laundry rooms can be booked three hours per occasion and maximum three occasions per every three weeks period.

- The two gyms, the tennis court and the ping-pong room can be booked between 9.00 a.m. to 9.00 p.m., one hour or maximum two hours per day.
- The sauna can be booked on the week-ends between 9.00 a.m. to 9.00 p.m., one hour per family and day.
- The banqueting room can be booked from one hour to one day.

A.2.3 User characteristics

The system will be used by a large group of different people; presumably most of them will have little or no computer experience. Therefore GUIs will be designed with special attention to the capability of the users and to ease of use of the touch screen.

A.2.4 Assumptions and dependencies

The software must work on the existing hardware.

A.2.5 Operational environment

The server will be a Windows 2000. A standard system for cipher coding and a TCP/IP protocol will be used in network communication. Figure A.1 shows the system design for the hardware in which the software shall be integrated:

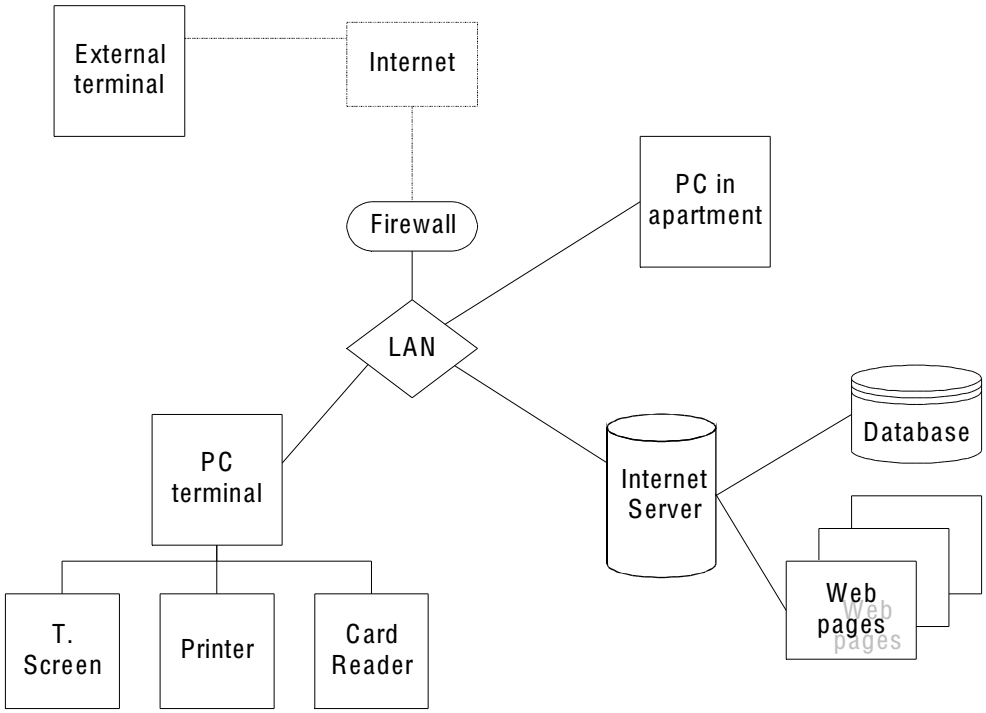


Figure A.1. Hardware system design.

A.3 Requirements

Server application
At installation of the server application, the maximum number of logged-in users allowed at the same time will be set to 100 users. Among them there will be no other priority order than “first logged-in first to book”. The connection to the database must be established.

Client application
The administrator installs the program on the terminal. When ready for use the general info site shall be displayed on the screen, from which the user may log in.

Logging in from a touch screen terminal
The user can log in by pressing the personal card to the card reader just below the touch screen. In addition the user has to press in a four digit personal code on the touch screen. If the numbers are incorrect the user will be informed of the error and asked to try again. If the numbers are correct the user will get access to the functions “book” or “change a booking”.

Logging in from an external terminal
The user can log in from any terminal with Internet access, in which case the personal card is not necessary. It is enough to write in the member number and the four digit personal code. The data will be checked with the database and if all is correct the user will see the same interfaces as on the terminals with a touch screen.

If a user is logged in but no function has been activated during 3 minutes the user will automatic be logged out as soon a function or a link is being activated on the current web page.

Booking requirements
The user shall be able to make a booking when the facility is allowed to be booked and not already booked. A survey calendar with the possible time periods for the object to be booked will be presented for the user who easily can choose the preferred period.

When a correct booking has been done the person will be asked if a receipt is desirable. If the “YES” button is pressed the receipt will be printed from the local printer. The receipt will include: full name, apartment number, the booked object, time period of the booking (day, hours), and the current date and time.

Changing a booking requirements
To make a change the user has to be logged-in and activate the "change a booking" button and select the appropriate facility. After the change the new information will be updated in the database. The booking change may only be performed by the user who made the booking or in the future (with the user's approval), by a person with the same apartment number. The administrator will of course be able to change any booking via the database.

GUIs requirements
Since a large and diverse mixture of people will use the system it is very important that the GUIs are clear and simple. Using simple words and small amounts of text on each page is one way to simplify. The buttons should be rather large to make it easy for the users to press them on the touch screen.

Database requirements
The relational database may be of the type Microsoft Access. It will store the data of the members who can use the system and the different facilities that are possible to book. Each facility has individual constraints, e.g. *how many times a user can book this object within a certain period*, that will be stored as well. A booking shall be stored a month after the booking date has expired to get some historic of the bookings.

Member data
- Card number (identity key)
- Name
- Four to five member number
- Four to six digit code
- Apartment number
- Sex
- Date of birth
- Family approval access (logical field or each approved member number)
- Other (text)
- Other (logical field)

Facility data
- Facility name (identity key)
- Maximum hours per day
- Maximum numbers of days per time period
- Other (text)
- Other (logical field)

Booking data
- Card number (identity key)
- Facility name (identity key)
- Date (identity key)
- Other (text)
- Other (logical field)

Performance requirements
The terminal connected to a touch screen shall have a processor with the minimum capacity of an Intel Pentium II 200MHz. The RAM shall not be less than 64MB. The terminal shall include a hard disc of minimum 2GB and be changeable to a Flash disc. It shall also be possible to connect a disc station, CD-ROM or some other store unit.

Environment requirements
- The software shall control and integrate the hardware such as the touch screen, ID-card reader, modem, and the receipt printer.
- The operating system of the internal terminals is a Windows NT 4.
- The server is a Windows 2000 or another general modern server.
- The GUIs will be web page based and the web browser will be an Explorer of optional version.
- TCP/IP is the network protocol to be used.

Reliability requirements
It is the supplier's responsibility to make sure the functions shall not take unreasonable long time to execute.

Safety requirements
All communication over the local network shall be cipher coded.

Appendix B

Architecture Design Document

Degree project

Web Based Booking System

By Sara Hvatz

Contents

B.1 Introduction 61
B.1.1 Purpose 61
B.1.2 Scope of the software 61
B.1.3 References 61
B.2 System design 62
B.2.1 Decomposition description 63
B.2.2 USE CASEs 63
B.2.2.1 Normal use cases 63
B.2.2.2 Exceptional use cases 65
B.3 Component description 67
B.3.1 Server 67
B.3.2 Client 68
B.3.3 Graphical User Interface 69
B.3.4 Database manager 70
B.3.5 Database 71

B.1 Introduction

The first part provides the purpose of the document itself and a short description of the product. The second part includes a developing process model, a class diagram of the software and, use cases. In the third part each component has a detailed description of the programs and their classes and it's functionality.

B.1.1 Purpose

The purpose of the Architecture Design document is to in detail define the Web Based Booking System's design and decomposition. The document describes the system's different programs and classes and how all interact. There may not be the same people who make the design who will develop the software therefore it is important that the programs and classes are detailed defined and unambiguous to enable many different software developers to implement the design.

B.1.2 Scope of the software

The booking system is including a database, a server, and web based terminals each connected to a touch screen, situated convenient for the users. As well there will be possible to connect to the server from an external web based terminal and have access to the system.

B.1.2.3 References

- The User Requirement Document
- There are several JavaScript tutorials on the web.
- *JavaScript for dummies* by Emily Vander Veer
- *JavaScript för webben* by Tom Negrino and Dori Smith
- *JavaScript Bible* (4th edition) by Danny Goodman
- *The Essential Client/Server Survival Guide* (2nd edition) by Robert Orfali, Dan Harkey and, Jeri Edwards

B.2 System design

Figure B.1 shows the Waterfall model that I will use when developing the prototype.

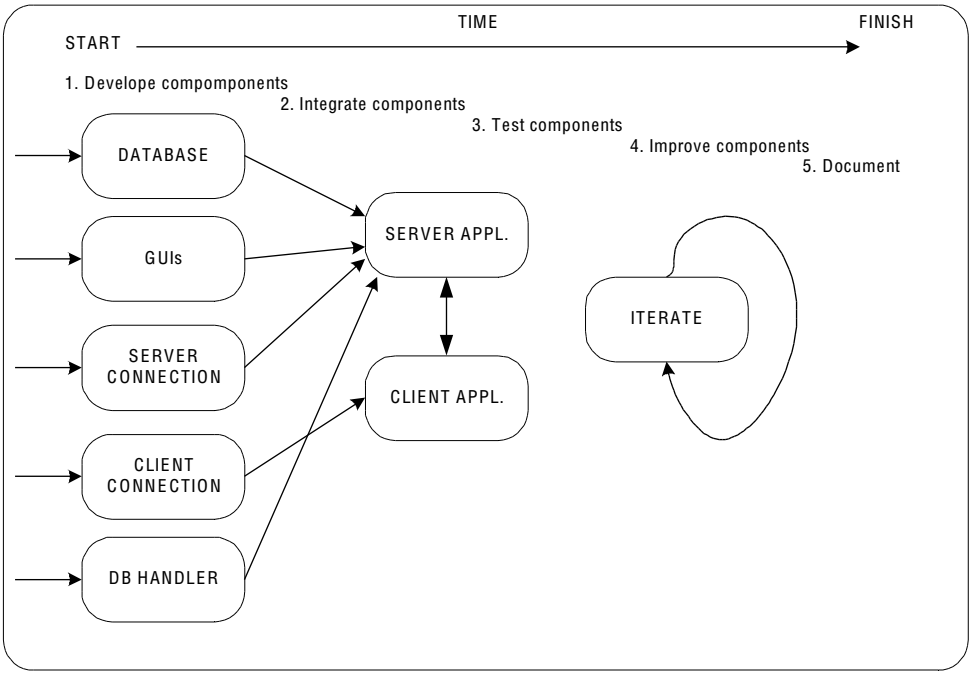


Figure B.20. The Waterfall model.

B.2.1 Decomposition description

Figure B.2 shows the state chart of the software.

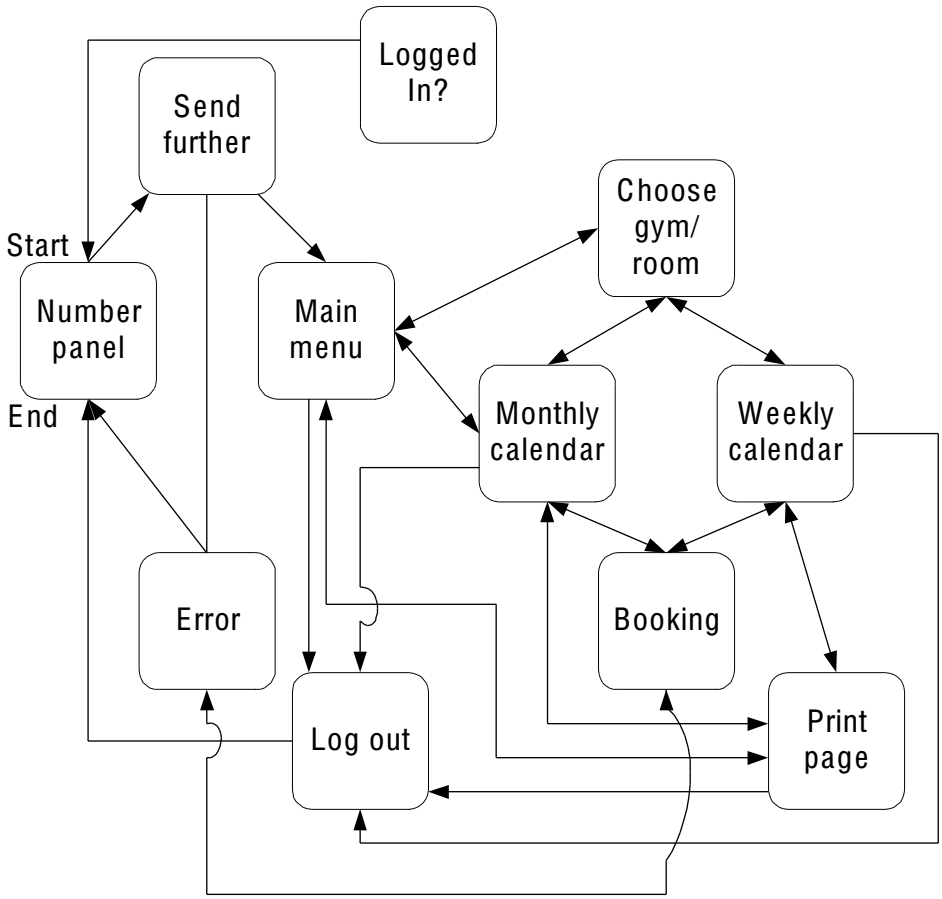


Figure B.2. State chart of WeBaBoS.

B.2.2 Use Cases

The use cases can be divided into normal and exceptional cases. A normal case is a case without any problems for example incorrect log-in code or maximum amount of users are already in the system. The exceptional cases describe all the cases (supposedly) when a problem might appear.

B.2.2.1 Normal use cases

Log In (from a local or external terminal)

- The user presses the “Log in” button from the general info page on the terminal.
- The client web browser asks the server if the user might log in.
- The server sends the “Log-In-Page”.
- The user presses the member card on the card reader and types in it’s personal code.

- The user data are sent to the sever.
- The server checks in the member table in the database if the card number exists, if the code is correct, and if the card number is unlocked.
- If everything is in order the server sends the main menu page and the user is logged-in.

Booking

- The user presses on one of the facilities from the main menu.
- The server sends back the facility's calendar page.
- The user presses on the available (green) time square which is linked to the booking function.
- The booking function is making sure the time has not been booked meanwhile and if not it checks the constraints whether the user may book that certain facility.
- If everything is allright the new booking is registered in the database and the server sends back the previous page with the new booking shown in the calendar.
- By pressing the "KLAR"-button on the calendar page the user can have a receipt of her bookings.

Print a receipt

- The user can get all of her bookings list and printed by pressing the "YES"-button on the print page.
- When the user presses the "YES"-button the print function gives order to the local printer to print the booking data that follows with the order.

Change a booking

- The user presses on one of the facilities from the main menu.
- The server send back the facility's booking page.
- The user presses on its own booked time slot, which is coloured (orange).
- The booking is deleted in the database.
- The server sends the previous page to the user and the time square is now available (green) again.

Log out

- The button "Log Out" is activated on the current page.
- The user is deleted from the logged_in table in the database.
- The server sends back the general information page.

B.2.2.2 Exceptional use cases

Impossible to log in because of to many users in the system

- The user presses the “Log in” button from the general info page on the terminal.
- The user types in the card number and code.
- The server sends a message to the user that he/she must try later because the maximum amount of users has been reached.

Impossible to log in because of card number’s inexistence

- The user presses the “Log in” button from the general info page on the terminal.
- The server sends the “Log-In-Page”.
- The user presses the member card on the card reader and types in it’s personal code.
- The user data are sent to the sever.
- The server checks in the member table in the database if the card number exists, if the code is correct, and if the card number is unlocked.
- If the card does not exist the server sends a page informing the user of the card’s inexistence and is asked to try again.

Impossible to log-in because of incorrect code.

- The user presses the “Log in” button from the general info page on the terminal.
- The server sends the “Log-In-Page”.
- The user presses the member card on the card reader and types in it’s personal code.
- The user data are sent to the sever.
- The server checks in the member table in the database if the card number exists, if the code is correct, and if the card number is unlocked.
- If the card exists but the code is incorrect the server sends a page informing the user of the code’s incorrectness and asks the user to try again.
- If the code is incorrect the second time the user is asked to contact the administrator or to try a third time. In addition it will be clear to the user that if the code is incorrect a third time the user will be unable to log in during X hours and only the administrator can unlock it.
- If the code is incorrect the third time the server puts a lock , with the current time, on the user which is saved in the member table in the database. Then it sends a message to the user that the lock is on which makes the user unable to log in during X minutes counting from the time the lock was set on.

Unable to book because the time slot was booked meanwhile.

- The user presses on one of the facilities from the main menu.
- The server send back the facility's booking page.
- The user presses on the available (green) time square which is linked to the booking function.
- The booking function checks if the time has been booked meanwhile and if not it checks whether the user may book that certain facility.
- If the time slot was booked meanwhile the user was choosing the server sends the error message page and informing the user of the double booking.

Unable to book because of some constraint was broken

- The user presses on one of the facilities from the main menu.
- The server send back the facility's booking page.
- The user presses on the available (green) time square which is linked to the booking function.
- The booking function checks if the time has been booked meanwhile and if not it checks whether the user may book that certain facility.
- If somehow this user was not allowed to book this certain facility the server application sends a page informing the user of the problem.

Automatic log out

- If a user is logged in but no function has been activated during 3 minutes the user will automatic be logged out as soon a function or a link is being activated on the current web page.
- The server sends the general information page and the user has been logged out.

B.3 Component description

B.3.1 Server

Type
A program.

The purpose
The server is the heart of the system. It will control the database manager and keeping track of the order of when the different web pages shall be executed on the clients.

Function
Here are some of the functions of the server.

- Be listening for signals from clients.
- After a correct log in has occurred, it will send the appropriate GUI to be displayed for the user.
- Get information from the database and send it further for the terminal together with the GUI.
- All the client/server communication will be cipher coded.
- It will carry out a booking or delete a booking.
- After a new booking (or change of a booking) of a certain facility has been inserted in the database (deleted) and confirmed all the current active web pages of that facility will be updated of the change.
- Before a booking can be inserted and confirmed it will always be a check if the time slot is still available.
- It will lock the user's card number out if the code has been incorrect three times.
- The server will log out the user automatically if the web page has not been activated in any way during 3 minutes.
- The administrator shall be able to manipulate the database directly from the server.

Subordinates

- DB Manager
- GUIs
- Client

Dependencies
Must have a database.

Interfaces

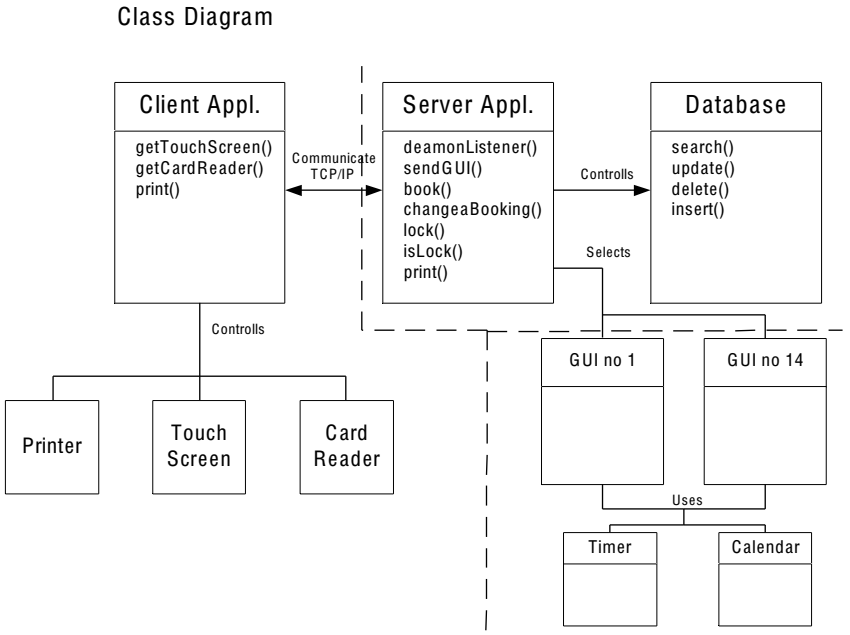


Figure 21. The Class diagram.

Resources
Needs a Windows 2000 server.

B.3.2 Client

Type
A client is a terminal with a web browser.

Purpose
The main purpose is to execute the web pages in the client's web browser. The internal clients also will have a small program installed on the internal computers to control the card reader.

Function
The client's function will be as mentioned to control the touch screen, the card reader and, the printer. It will provide the web page with data from its parts. The client will have no direct contact with the server though all contact will go throw the web pages.

Subordinates
No.
Dependencies
It has to control the card reader, touch screen and, the printer.
Interfaces
No.

B.3.3 Graphical User Interface

Type

Each web page can be defined as one class.

Purpose

The GUIs work as the mediator between the user and the system. Through a user interface the user is able to use the system's functions. These different GUIs are necessary for the system to interact with the user.

Function

The GUIs will be web based and dynamic and therefore written in HTML and JavaScript. The pages will be executed on the clients by the web browser Microsoft Internet Explorer 5. The pages will be designed to fit the touch screen size (21 * 29 cm) and by default without toolbars and status bar.

The function buttons will be activated by pressing the touch screen gently or by a mouse click to make it possible to use the same pages for both internal and external terminals.

Once the user has logged-in each web page must use a timestamp to be able to inform the server of user inactivity during 3 minutes and because of this the server will perform an automatic log out.

A calendar will be used for display free-for-booking time slots to the user. Time slots from yesterday or older can be displayed but indicated inactive. How detailed the calendar will appear to the user depends of the facility. The most detailed calendar includes:

- Year
- Month
- Day
- Hour

As well as the use cases the GUIs can be divided into normal and exceptional case web pages depending on in which case they will appear to the user.

The normal case pages:

- Information page
- Log in
- Main menu (choose a facilities)
- Booking / change a booking
- Print

The exceptional case pages:
- Unacceptable card or member number
- Invalid code
- The card is locked
- Too many users in the system
- The time slot became booked
- You are not allowed to book more time slots
- No server connection

Subordinates
No.

Dependencies
No.

Interfaces
The server gives signals to send the appropriate page together with correct data from the database to the client where the page gets browsed and displayed for the user. The browser reads the interactions of the user and the new data are sent back to the server.

Resources
Needs a web browser that can execute JavaScript.

B.3.4 Database manager

Type
A class.

Purpose
The database will contain information about the excepted users, administrators and facilities of the system. It will as well keep information of the booking relations.

Function
In some cases there might be useful to get some historical facts and therefore all the booking data will be saved up to a month after the date of the booking thereafter deleted.

The database manager will use the Structured Query Language (SQL) to manipulate the database. Here are some typical SQL methods:
- search()
- update()
- delete()
- insert()

Subordinates
No.

Dependencies
No.

Interfaces
- The server uses the database manager's methods to manipulate the database.
- The web pages uses the database manager's methods for example when the user tries to log in.

Resources
The database will be a MySQL database.

B.3.5 Database

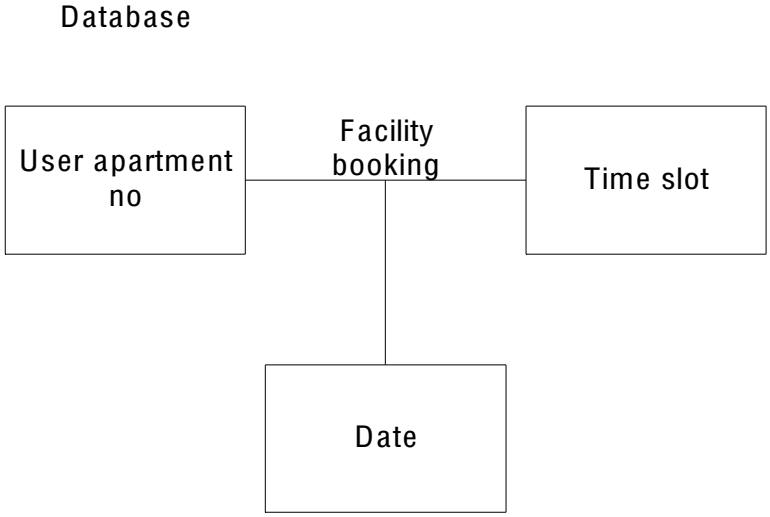


Figure 22. The Database.

Here are the relations in the database and their fields.

Member data:
- Card number (primary key)
- Name
- Four to five member number
- Four to six digit code
- Apartment number
- Sex
- Date of birth

- Family approval access (logical field or each approved member number for future improvement)
- Extra attribute (text)
- Extra attribute (logical field)

Facility data:
- Facility name (primary key)
- Number of facilities of same type
- Maximum sessions per time period
- Maximum numbers of days per time period
- Extra attribute (text)
- Extra attribute (logical field)

Facility booking :
- Time slot (primary key)
- Date (primary key)
- Apartment number (primary key)
- Extra attribute (text)
- Extra attribute (logical field)

Appendix C

The Software User Manual for the Web Based Booking System

Degree project

Web Based Booking System

By Sara Hvatz

Contents

C.1 Numberpanel.php 75
C.1.1 Images 75
C.2 Sendfurther.php 75
C.3 Mainmenu.php 76
C.3.1 Images 76
C.4 Chooseroom.php 78
C.4.1 Images 79
C.5 Choosegym.php 79
C.5.1 Images 79
C.6 Monthcalendar.php 79
C.7 Basecalendar.inc 80
C.7.1 Images 80
C.8 Ex2_Style.css 80
C.9 Ex2(cal2)_Style.css 81
C.10 Weekcal.php 81
C.11 Weekcalendar.inc 81
C.12 Week_style.css 81
C.13 Tobook.php 82
C.14 Booking.inc 82
C.15 Printmenu.php 83
C.15.1 Images 83
C.16 Loggedin.inc 83
C.17 Logout.php 84
C.18 Error.php 84
C.19 Database structure 86
C.20 Card reader program 88
C.21 Web page network 89

C.1 Numberpanel.php

Purpose
The page from where the users can log in.

Input
The user's card/member number and code.

Output
The user's card/member number and code to the Sendfurther.php-page sent by the method "post".

Error handling
None.

C.1.1 Images

The images included in the set of number buttons
(made in Jasc Software: Paint Shop Pro 7)
Name: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, rensa, klar
Format: .jpg
Size: (each) 110 * 110 pixels
Color: gold metallic and buttonized
Font size: 60, 14 for rensa och klar

The text square
(made in Macromedia: Fireworks MX)
Name: login
Format: .gif
Size: 600 * 100 pixels
Color: #66CCFF (blue)
Font size: 30

C.2 Sendfurther.php

Purpose
To check if the user has logged in correctly and that the system has not too many users inside the system already.

Input
The user's card/member number and code.

Output
Id to the mainmenu-page.

Error handling
Generates these error messages to error.php:

- “Kortnummer eller medlemsnummer fattas, försök igen!”
- ”Kod fattas, försök igen!”
- ”Den här medlemmen är redan inloggad!”
- ”Kortet är spärrat kontakta en administratör för upplåsning!”
- ”Ditt kortnummer eller kod finns inte med i databasen, försök igen!”
- ”Systemet är överbelastat var vänlig försök senare!”

C.3 Mainmenu.php

Purpose
To present the facilities.

Input
Id

Output
Id, facility name to the monthcalendar.php, choosegym.php or chooseroom.php

Error handling
Sends the user to numberpanel.php if the user has not logged in.

C.3.1 Images

The text square
(made in Macromedia: Fireworks MX)
Name: login
Format: .gif
Size: 600 * 100 pixels
Color: #66CCFF (blue)
Font size: 30

The facility buttons
(made in Macromedia: Fireworks MX)
Name: gym, tvättstuga, bastu, festlokal, tennisbana, pingislokal
Format: .gif
Size: 180 * 80 pixels
Color: #00FFCC (green)
Font size: 24 (bold)

The out curved buttons
(made in Macromedia: Fireworks MX)
Name: logout, all
Format: .gif
Size: 160 * 100 pixels
Color: #66CCFF (blue) Filters>>Eye Candy>>Bevel Boss
Basic:
Bevel Width: 6.00 pixels
Bevel Height Scale: 50
Smoothness: 60
Lightning:
Direction: 135°
Inclination: 80°
Font size: 20

C.4 Chooseroom.php

Purpose
To chose which laundry room to access.

Input
Id, facility name

Output
Id, facility name, mes, room, and numday to the weekcal.php.

Error handling
Sends the user to numberpanel.php if the user has not logged in.

C.4.1 Images

The text square
(made in Macromedia: Fireworks MX)
Name: valjgym
Format: .gif
Size: 600 * 100 pixels
Color: #66CCFF (blue)
Font size: 30

The facility buttons
(made in Macromedia: Fireworks MX)
Name: stuga1, stuga2, stuga3, stuga4, stuga5, stuga6
Format: .gif
Size: 180 * 80 pixels
Color: #00FFCC (green)
Font size: 24 (bold)

The out curved buttons
(made in Macromedia: Fireworks MX)
Name: logout, goback, all
Format: .gif
Size: 160 * 100 pixels
Color: #66CCFF (blue) Filters>>Eye Candy>>Bevel Boss
Basic:
Bevel Width: 6.00 pixels
Bevel Height Scale: 50
Smoothness: 60
Lightning:
Direction: 135°
Inclination: 80°
Font size: 20

C.5 Choosegym.php

Purpose
To chose which gym to access.

Input
Id, facility name

Output
Id, facility name, mes, room, and numday to monthcalendar.php.

Error handling
Sends the user to numberpanel.php if the user has not logged in.

C.5.1 Images

The text square
(made in Macromedia: Fireworks MX)
Name: valjgym
Format: .gif
Size: 600 * 100 pixels
Color: #66CCFF (blue)
Font size: 30

The facility buttons
(made in Macromedia: Fireworks MX)
Name: gym1, gym2
Format: .gif
Size: 180 * 80 pixels
Color: #00FFCC (green)
Font size: 24 (bold)

The out curved buttons
(made in Macromedia: Fireworks MX)
Name: logout, goback, all
Format: .gif
Size: 160 * 100 pixels
Color: #66CCFF (blue) Filters>>Eye Candy>>Bevel Boss
Basic:
Bevel Width: 6.00 pixels
Bevel Height Scale: 50
Smoothness: 60
Lightning:
Direction: 135°
Inclination: 80°
Font size: 20

C.6 Monthcalendar.php

Purpose
To receive and send further the variables to the function “displayMonth” in the Basecalendar.inc-class.

Input
Id, facility name, room, mes, numday,

Output
Id, facility name, room, mes, and numday to basecalendar.inc.

Error handling
Sends the user to numberpanel.php if the user has not logged in.

C.7 Basecalendar.inc

Purpose

To create and display the monthly calendar with start on a Monday.

Input

Id, facility name, room, mes, numday,

Output

Link to last/next month with Id, facility name, room, mes, numday. Link to another day withId, facility name, room, mes, numday. Link to tobook.php with Id, calendar, facility name, room, mes, numday, date, time slot.

Error handling

None.

C.7.1 Images

The out curved buttons (above)

(made in Macromedia: Fireworks MX)
Name: nextmonth, lastmonth
Format: .gif
Size: 180 * 48 pixels
Color: #66CCFF (blue) Filters>>Eye Candy>>Bevel Boss
Basic:
Bevel Width: 6.00 pixels
Bevel Height Scale: 50

The out curved buttons (below)

(made in Macromedia: Fireworks MX)
Name: logout, goback, done
Format: .gif
Size: 160 * 100 pixels
Color: #66CCFF (blue) Filters>>Eye Candy>>Bevel Boss
Basic:
Bevel Width: 6.00 pixels
Bevel Height Scale: 50
Smoothness: 60
Lightning:
Direction: 135°
Inclination: 80°
Font size: 20

C.8 Ex2_Style.css

Purpose

Design sheet for the Basecalendar.inc desing. Iincludes size-, color-, and font-setups.

C.9 Ex2(cal2)_Style.css

Purpose

Design sheet for the Basecalendar.inc desing. Iincludes size-, color-, and font-setups.

C.10 Weekcal.php

Purpose

To receive and send further the variables to the function “displayMonth” in the Weekcalendar.inc-class.

Input

Id, facility name, room, mes.

Output

Id, facility name, room, and mes to weekcalendar.inc.

Error handling

Sends the user to numberpanel.php if the user has not logged in.

C.11 Weekcalendar.inc

Purpose

To display the facility’s bookings in a weekly calendar with start on the current week (Monday to Sunday).

Input

Id, facility name, room, mes.

Output

The calendar.

Error handling

None.

C.12 Week_style.css

Purpose

Design sheet for the Weekcalendar.inc desing. Iincludes size-, color-, and font-setups.

C.13 Tobook.php

Purpose
To receive and send further the variables to the function “bookOrUnbook” in the Booking.inc-class and after go back to the same page and refresh it so that the change gets visible.

Input
Id, facility name, room, calendar, mes, numday, date, time spot.

Output
Id, facility name, room, calendar, mes, numday, date, and time spot to booking.inc.

Error handling
Sends the user to numberpanel.php if the user has not logged in.

C.14 Booking.inc

Purpose
To book or change a booking if the constraints are fulfilled.

Input
Id, facility name

Output
No.

Error handling
Generates these error messages to error.php:

- “Det misslyckades att boka!”
- ”Du måste avboka en annan tid för att kunna boka en ny!”
- ”Det misslyckades att avboka!”
- ”Tiden tillhör lägenhet X!”
- ”Något gick fel i bookorunbook-funktionen!”

C.15 Printmenu.php

Purpose

To list the user name, apartment number, the facility name, and time period of the booking (day, hours) for the local print to print out if the user presses the “YES”-button.

Input

Id, facility name.

Output

No.

Error handling

Sends the user to numberpanel.php if the user has not logged in.

C.15.1 Images

The out curved buttons

(made in Macromedia: Fireworks MX)
Name: logout, goback, headmenu, yes
Format: .gif
Size: 160 * 100 pixels
Color: #66CCFF (blue) Filters>>Eye Candy>>Bevel Boss
Basic:
Bevel Width: 6.00 pixels
Bevel Height Scale: 50
Smoothness: 60
Lightning:
Direction: 135°
Inclination: 80°
Font size: 20 (“yes” has font size 30)

C.16 Loggedin.inc

Purpose

The page checks if the user has logged in and if the timestamp of when it was last updated has not expired. If the user is logged in and if the timestamp has not expired the timestamp is being updated.

Input

Id.

Output

True or false.

Error handling

None.

C.17 Logout.php

Purpose
To log out the user, by deleting the card number from the *logged_in* table. When the user has been logged out the page will send the user to the log-in page (numberpanel).

Input
Id.

Output
Numberpanel.php.

Error handling
Generates this error message to error.php:

- "Det misslyckades med utloggningen!"

C.18 Error.php

Purpose
To print out the error message which is the input to this page.

Input
Message string.

Output

- Id, mes, numday, facility, and room to monthcalendar.php,

- Id, mes, numday, facility, and room to weekcal.php,

history.back() which is a predefined function and goes to the previous page.

Error handling
Sends the user to numberpanel.php if the user has not logged in.

C.18.1 Images

The out curved button
(made in Macromedia: Fireworks MX)
Name: goback
Format: .gif
Size: 160 * 100 pixels
Color: #66CCFF (blue) Filters>>Eye Candy>>Bevel Boss
Basic:
Bevel Width: 6.00 pixels

Bevel Height Scale: 50
Smoothness: 60
Lightning:
Direction: 135°
Inclination: 80°
Font size: 20

C.19 Database structure

These tables are included in the database:
member (cardnumber char(20) default '' NOT NULL,
name char(40) default '' NOT NULL,
member_number char(20) default '' NOT NULL,
code char(10) default '' NOT NULL,
ap_number char(10) default '' NOT NULL,
sex char(8) default '' NOT NULL,
attempts char(20) default '' NOT NULL,
timestamp char(20) default '' NOT NULL,
extra_1 char(20) default '', extra_2 char(20) default '',
PRIMARY KEY (cardnumber))

facility_data (fac_name char(20) default '' NOT NULL,
fac_no char(20) default '' NOT NULL,
max_slots char(20) default '' NOT NULL,
max_days char(20) default '' NOT NULL,
extra_1 char(20) default '',
extra_2 char(20) default '',
PRIMARY KEY (fac_name))

Tvättstuga_1 (date char(20) default '' NOT NULL,
ti_slot char(10) default '' NOT NULL,
ap_number char(10) default '' NOT NULL,
PRIMARY KEY (date,ti_slot))

general_data (max_inlogged char(10) default '100' NOT NULL,
automatic_logout char(10) default '3' NOT NULL,
max_attempts char(10) default '3' NOT NULL,
lock_time char(10) default '60' NOT NULL,
extra_1 char(10) default '',
extra_2 char(10) default '',
PRIMARY KEY (max_inlogged))

logged_in (cardnumber char(20) default '' NOT NULL,
ti_stamp char(12) default '' NOT NULL,
PRIMARY KEY (cardnumber))

The *member* table stores information about the residents of the building. Everyone shall have, according to Robert Frick, who is the orderer of the system, a personal card, member number, and code with which they can log in to the system. The "attempt" variable represents the number of times a person may try to log in with an incorrect code before the card number will be locked. The timestamp in the table is the exact time when the lock was activated.

The *facility_data* has information about the facility's constraints, such as how many time slots, "max_slots", a user can book during a period, and how many days, counted from the current day until the variable max_days, the period includes for that particular facility. The calendar in the system depends on the maximum number of days the facility can be booked in advance. Exclusively only the days within the period are linked to the booking application and can therefore be booked.

The *Tvättstuga_1* table is one of the facilities that can be booked. In this system there are 6 laundry rooms but each room has its own table in the database, just like the other facilities. Each of these facilities includes a date and a time slot, which together make the primary key. This is a reasonable solution since only one family at a time can use the facility at that particular day and time.

The *general_data* table stores data that has to do with the special requirements. The first column represents the maximum number of logged-in users allowed at the same time. It is by default set to 100 users because that is one of the requirements in the URD. The second column value in the table is the time, measured in minutes, before the automatic logout is activated if no buttons or links have been activated during these minutes. The "lock-time" variable is the time before a lock on a card number is unlocked again. The lock is set by default to be locked during one hour (60 minutes) if the lock has been activated.

To provide security, the system's web pages are inaccessible unless the user logs in. When a log-in is being successfully carried out the user's card number will be stored in the *logged_in* table together with the timestamp (ti_stamp) of when the user logged in. Each time a new web page is entered the logged_in table is checked. If the user's card number is in it and if the timestamp has not exceeded the "automatic_logout"-time the user accesses the page, otherwise the log-in page is shown.

C.20 Card reader program

Keyboard.zip

The software includes a separate program that is installed on the internal clients. The program is always listening for messages from the card reader and transfers the characters to the active field on the web page. The program simulates keyboard presses. The software includes a function that prints a page to the default printer of that specific computer where the user has logged in on.

C.21 Web page network

This picture represents the network of main web pages the WeBaBoS is built on. Every page is connected to the Logged-in?-square (the arrows are not set out because it would make the picture too messy) which represents that a user who has not logged in will automatic be transferred to the numberpanel-page. The arrows show to which state (page) the user can go to from the current state (page).

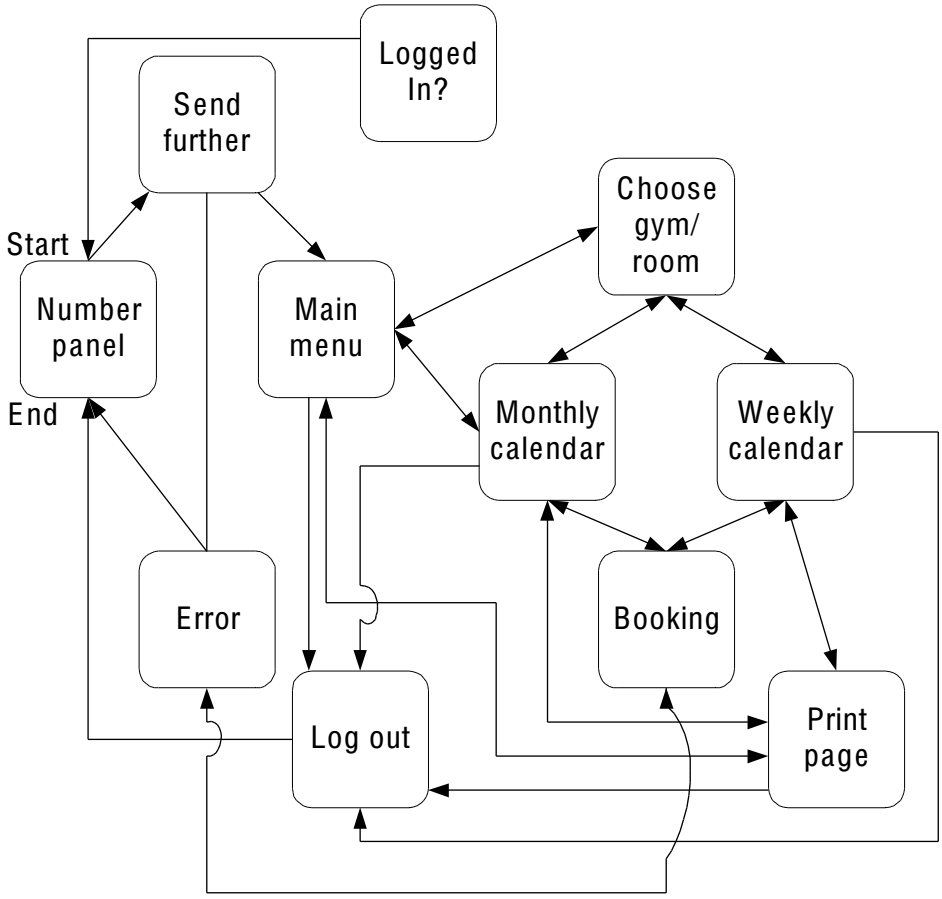


Figure C.1. Web page network.

Appendix D

The Software Transfer document for the Web Based Booking System

Degree project

Web Based Booking System

By Sara Hvatz

Contents

D.1 Sunny day scenarios tests 91
D.2 Rainy day scenarios tests 94
D.3 Different web browser 95
D.4 Server crash 95
D.5 Database crash 95

D.1 Sunny day scenarios tests

The sunny day scenarios are the cases in which the functions work according to the requirements. Here are the requirements taken from the URD:

1. **All communication over the local network shall be cipher coded**. - The code has **not** been cipher coded because one needs to buy a special certification from Microsoft to be able to deactivate the existing Internet Explorer cipher program. It was decided that it would be bought only when the prototype will be a product later on.
2. **The server shall be connected to a database**. - The database that works with the server is the MySQL. An open source database server that is widely used all over the world.
3. **The software shall control and integrate the hardware such as the touch screen, ID-card reader, modem, and the receipt printer**. - The software does not have to control the touch screen since the computer itself take care of the functions of clicking and moving the cursor. The software includes a separate program that is installed on the internal clients. The program is always listening for messages from the card reader and transfers the characters to the active field on the web page. The program simulates keyboard presses. The software includes a function that prints a page to the default printer of that specific computer where the user has logged in on.
4. **The software shall be as general as possible to facilitate future functions adding for e.g. "open a door" function and a camera connection**. - The software is general in that sense that all the important data is stored in the database, such as the member data, booking data, and facility constraints. If there are facilities or members that are to be added in the system the only thing the administrator has to do is to add those in the database not in the software itself. It would also be easy to add for example an "open a door" function where a small application can check in the database to find out if that particular card number is allowed access.
5. **General information page with a log in link as the front page**. - The general information page has **not** been developed because of time shortage.
6. **The facilities have different booking constraints**. - The facilities have different booking constraints, meaning they have different maximum time slots per period and different length of the periods. The calendar does not allow bookings on days that are outside of this period. The period always start from "today" and ends on the maximum of days in the period, stored in the facility data table.
7. **The user can log in by pressing the personal card to the card reader just below the touch screen. In addition the user has to**

press in a four digit personal code on the touch screen. - The scenario has been tested with a card and code. The functions work as they should.

8. **The user can log in from any terminal with Internet access by writing in the member number and the four digit personal code**. - Yes, a user can successfully log in from any terminal with Internet access. However if the web browser does not support JavaScript all of the booking system's functions will not be functional.
9. **The user shall be able to make a booking when the facility is allowed to be booked and if that time slot is not already booked. A survey calendar with the possible time periods for the object to be booked will be presented for the user who easily can choose the preferred period**. - Yes, the user can book an available time slot, represented as a green square with the text "free".
10. **The user shall have the possibility to get a receipt of all the bookings. If the "YES" button is pressed on the receipt page the receipt will be printed from the local printer. The receipt will include: full name, apartment number, the booked object, time period of the booking (day, hours)**. - The user can get a receipt with all the current bookings, which hold her apartment number of either all the facilities or only the facility from which page he/she pressed the button "KLAR" (ready).
11. **The GUIs will be web page based and the web browser will be an Explorer of optional version**. - The GUIs are web based and they are designed after the Internet Explorer Web browser.
12. **The GUIs shall be clear and simple and the buttons should be rather large**. - The GUIs are clear and simple, even for beginners (they are tested on beginners), with large buttons and squares.
13. **The database shall include**:

Member data

- Card number (identity key)
- Name
- Four to five member number
- Four to six digit code
- Apartment number
- Sex
- Date of birth
- Other (text)
- Other (logical field)

Facility data

- Facility name (identity key)
- Number of facilities of the same kind
- Maximum time slots
- Maximum numbers of days per time period
- Other (text)
- Other (logical field)

Booking data

- Card number (identity key)
- Facility name (identity key)
- Date (identity key)
- Apartment number
- Other (text)
- Other (logical field)

The database includes:

member (cardnumber char(20) default '' NOT NULL,
name char(40) default '' NOT NULL,
member_number char(20) default '' NOT NULL,
code char(10) default '' NOT NULL,
ap_number char(10) default '' NOT NULL,
sex char(8) default '' NOT NULL,
attempts char(20) default '' NOT NULL,
timestamp char(20) default '' NOT NULL,
extra_1 char(20) default '', extra_2 char(20) default '',
PRIMARY KEY (cardnumber))

facility_data (fac_name char(20) default '' NOT NULL,
fac_no char(20) default '' NOT NULL,
max_slots char(20) default '' NOT NULL,
max_days char(20) default '' NOT NULL,
extra_1 char(20) default '',
extra_2 char(20) default '',
PRIMARY KEY (fac_name))

Tvättstuga_1 (date char(20) default '' NOT NULL,
ti_slot char(10) default '' NOT NULL,
ap_number char(10) default '' NOT NULL,
PRIMARY KEY (date,ti_slot))

general_data (max_inlogged char(10) default '100' NOT NULL,
automatic_logout char(10) default '3' NOT NULL,
max_attempts char(10) default '3' NOT NULL,

lock_time char(10) default '60' NOT NULL,
extra_1 char(10) default '',
extra_2 char(10) default '',
PRIMARY KEY (max_inlogged))

logged_in (cardnumber char(20) default '' NOT NULL,
ti_stamp char(12) default '' NOT NULL,
PRIMARY KEY (cardnumber))

D.2 Rainy day scenarios tests

The rainy day scenarios are the cases when an error message will be generated. Here are the "negative" scenarios taken from the URD:

1. **Impossible to log in because of too many logged in users in the system**. - The constraint has been tested with maximum of 3 logged-in users at the same time. When the fourth tried to log in an error message was shown for the user that informs the user of the system overloading and is asked to try later.
2. **Impossible to log in because of wrong card number**. - The scenario has been tested with incorrect card number and an informative error message is shown to the user.
3. **Impossible to log in because of incorrect code**. - The scenario has been tested with incorrect code and an informative error message is shown to the user.
4. **If an in-logged user has not activated a button during 3 minutes the user will automatic be logged out.** - If no function has been activated during 3 minutes the users will be automatically logged out as soon as someone tries to click on a function or a link.
5. **Unable to book because of some facility constraint**. - The calendars make sure that the users cannot book the facility too far in advance, by not linking the days that are not in the facility's booking period. If a user tries to book more time slots than the facility constraint allows an error message will inform the user that he/she has to change another booking to be able to book a new one.
6. **Unable to book because the time was booked meanwhile**. - When two (or more) users try to book the same time slot at the same moment has tested. The database manager makes sure that two primary keys cannot be inserted and therefore one of user gets an error message generated by the WeBaBoS informing of the fact that the time slot was booked meanwhile by someone else. From the error message page the user can go back to the previous page, which will be

refreshed and the booked time slot will show the apartment number of the user who booked the time slot.

7. **Unable to change another user's booking**. - A booking can be changed only by a person who lives in the apartment which number the booking holds. If a user tries to change another user's booking an error message will inform the user that the time is occupied.

D.3 Different web browser

The system is designed to work on the Internet Explorer but has been tested on Netscape and Opera as well. Apparently none of these two browsers supported JavaScript, which is used in, for example, the log in page. The web pages were deformed because of different standard resolutions and the colours were painted differently or some of them not painted at all. The "YES"-button on the print page was out of function in both browsers because of the lack of JavaScript support and therefore it was not possible to print the bookings.

D.4 Server crash

A server crash has been simulated, by turning of the server in the middle of a use session. The user got error messages generated by the Internet Explorer web browser informing of the connection error. The bookings that were booked before the crash were safely saved in the database. The user could not log in again directly after the crash since the card number was still in the *logged_in* table so he/she had to wait until the automatic logout time had expired.

D.5 Database crash

A database crash was simulated, by turning of the database server in the middle of a session. The WeBaBoS generated database connection errors and the user was told to contact an administrator. The new bookings that the user had done before the crash were safely saved in the database.